SecureXML Digital Signature & Encryption

User's Guide and Programmer's Reference

Version 2.5.148.46
Dated: June 26, 2007

Infomosaic Corporation
3005 Martin Meadows Court
Ellicott City, MD 21042
Phone: (703) 953-2398
http://www.infomosaic.com

Copyright Notice

```
/*****************************************************************************************
 * Copyright: (c) 2000-2005 Infomosaic Corporation.
 * All rights reserved.
 * It is violation of international law to use this code without proper written authorization
 * and license agreement from Infomosaic Corporation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL INFOMOSAIC CORPORATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ================================================================
 *
 *
 *
 *****************************************************************************************
 */
```

Table of contents

Release Information

Version 2.5.148.46

New Features

- None

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- The API parameter type of unsigned long prevented VB6 to use CanonicalizationMethod. This has been changed to long allowing VB6 application to set CanonicalizationMethod property as needed.

Other Changes

- None.

Version 2.5.147.45

New Features

- None

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- An issue related to canonicalization of XML elements has been fixed.

Other Changes

- EncryptFile and DecryptFile methods have been internally modified to support encryption and decryption of large files without requiring large amount of memory. This is achieved by encrypting/decrypting in chunks. Previously entire file was encrypted//decrypted at once, which require large amount of memory for large files. Also these methods no longer do based64 encode/decode of the file data.

Version 2.5.146.44

New Features

- None

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- An authentication issue while fetching CRLS from the NIST LDAP servers has been fixed.

Other Changes

- None

Version 2.5.145.44

New Features

- Sun Solaris on SPARC is now supported as a server. Java is the only language supported in Solaris.
- Support for Certificate Selection Dialog box in Purtuguese has been added.
- Support for <Signature> elements with no Id attribute is now supported.

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- In some cases a few temporary files created by SecureXML were not deleted during object destruction. This has been fixed.
- Small memory leaks under the following use cases have been fixed:
    - JITC Compliant policy enforcement during certificate validation.
    - Split signing, if a signing certificate was not selected before digest calculation.
    - Certificate validation
    - Calls to APIs which return an array of strings e.g. getAuthorityConstrainedPolicy, getUserConstrainedPolicy, getUsedCRLList etc.
    - LDAP based CRL access.
- Acess to URIs passed to AttachedObjects containing Unicode characters lead to an error. Now Unicode characters in URIs passed to AttachedObjects is processed correctly.
- An access violation error on the client side during a call to SignSignedInfoDigest method has been fixed. This error showed up on certain machines when a signing certificate was not selected before digest calculation during the split signing operation.
- Fixed problem related to accessing the private from a few vendor specific CSPs.

Other Changes

- None

Version 2.5.144.44

New Features

- None

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- A bug related to accessing Netscape certificate store certificates when Netscape browser was also engaged in cryptographic operations such as SSL or PKI key generation, has been fixed.

Other Changes

- The file NetscapeCert.dll has been replaced with NetscapeCert.exe

Version 2.5.143.43

New Features

- SNTP based time stamping support has been added.
- Ability to specify the time stamp format has been added.
- Ability to require time stamp for successful creation of a signature has been added.

Following object methods have been added:

- None

Following object properties have been added:

- TimeStampFormat
- TimeStampCritical

Bug Fixes

- A bug related to accessing Netscape certificate store certificates has been fixed.

Other Changes

- Whenever time stamping is enabled during signature creation, the following two signature properties are added to the signature created:
    - TimeStampFormat (Values can be 0, 1 or 2)
    - TimeStampProtocol (Values can be either SNTP-RFC-1305 or DAYTIME-RFC-867)

Version 2.4.142.42

New Features

- SecureXML Java Applet now supports both ActivCard Gold for CAC 2.2/3.0 and Litronic NetSign CAC 4.2 smart card reader middleware.
- Ability to read CN and email address from the CAC has been added

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- LDAP based CRL fetch now processes all records returned by the LDAP server. Previously it processed only the first record.

Other Changes

- None.

Version 2.4.141.41

New Features

- In addition to English, certificate selection dialog box is now also available in Japanese, French, Spanish, German and Hungarian.
- Floating client license for a given licensed server is now supported.
- SecureXML now provides ability to prepare XML envelope for both Attached and Detached signed objects making the end user application completely free from XML parsing/processing when signing non-XML data.
- All signature verification APIs are now capable of verifying detached signatures.
- Setting DetailedVerificationFlag to 2, now does not prepare signed objects for access after signature verification. This reduces the total memory requirement during signature verification by more than 50% and makes verification 20% faster. This feature is most useful when a signature produced over large detached signed data is being verified.
- Now signer certificate information can be added during a call to ApplySignatureValue. This allows for the client side to delay the certificate selection until after the digest calculation has been completed on the server side.
- A SecureXML Java Applet is now available for use on the client side. It can be downloaded and installed without having administrative privilege on the client machine and hence makes client side deployment to a large number of machines and their subsequent maintenance very easy.
- Certificate selection dialog box is now supported for both Java standalone applications and Java applets.
- Support for US DoD Common Access Card (CAC) demographic information access is now available through the SecureXML Java Applet. The end user must have ActivCard Gold 2.2 smart card reader software installed in order to read the demographic information using the Infomosaic SecureXML Java Applet.

Following object methods have been added:

- None

Following object properties have been added:

- FloatingLicense
- Language

Bug Fixes

- None.

Other Changes

- This version of SecureXML requires a new license file. Your old license file will not work. In order to get your new license file please visit http://www.infomosaic.net/InstallLicense and follow directions. You must be under maintenance in order to obtain a new license file.

Version 2.3.140.40

New Features

- The certificate selection window and the CertificateCount property both now exclude certificates which do not have the digital signature usage enabled.

Following object methods have been added:

- None

Following object properties have been added:

- None

Bug Fixes

- A few file access methods failed when the input file path contained white spaces or Unicode characters. This problem has been fixed. This problem was discovered during internal testing.

Other Changes

- Due to security issues, a number of methods which write to the disk now check for the client type and do not allow writing to a known location when invoked from a web browser. They create a temporary file instead, which is deleted during object destruction. In other words, as the user browser switches to another page the temporary file is automatically deleted. Hence the data saved from a browser page must be consumed in the same page. The following is a list of methods which have been impacted:

    - SaveXMLStr
    - SaveXMLByteArray
    - Base64DecodeBufferToFile
    - Base64DecodeFileToFile
    - Base64DecodeByteArrayToFile
    - Base64EncodeStrToFile
    - Base64EncodeByteArrayToFile
    - SecureXMLVerifyFileToFile

Version 2.3.139.39

New Features

- Proxy server use for OCSP based certificate validation is now supported
- SecureXML is now supported on Linux in addition to Windows. The APIs for Linux are the same as those for Windows.
- Exclusive canonicalization is now supported. Set the CanonicalizationMethod property to 2 or 3 in order to enable this feature. Please note that enabling exclusive canonicalization will make operations slightly slower.

Following object methods have been added:

- None

Following object properties have been added:

- ProxyHost
- ProxyPort
- ProxyUserName
- ProxyPassword
- InclusiveNamespacePrefixList

Bug Fixes

- The browser crashed if a user clicked the OK button after clicking in the white space in the certificate selection window.
- Adding OCSP Text Response to the current signature led to a crash for certain certificates/OCSP Server combinations.

Other Changes

- SecureXML Java Class infomosaic.securexml.Signature no longer uses jacob.jar and jacob.dll files and hence it is no longer sensitive to JDK versions. It uses SignatureL.dll instead of XMLSign.dll. SecureXML now works with all JDKs 1.2 and higher. Java programmers would need to call the Signature class destroy() method to free resources used by the object. Failure to do so will lead to memory leaks since JVM can't release the memory allocated by native code.
- The dependency of XMLSign.dll on gdiplus.dll has been encapsulated away into a separate dll called SigWinImage.dll, thereby allowing easy deployment of XMLSign.dll using a cab file. Earlier such deployments failed if the machine either didn't have gdiplus.dll or if it was not in PATH.
- A signed cab file securexml.cab is now included with the SDK, which allows for easy client side deployment.

Version 2.2.138.39

New Features

- CAM Validation Transaction Id is now included as a signature property whenever CAM is used for certificate validation during signature creation. CAM Validation Transaction Id is also available as a property for the client program.
- A more flexible form based licensing where schema match is not enforced is now supported.
- The client program can now specify if incomplete Signature elements should be ignored during signature verification.
- The client program can ask for a specific signature to be verified when the input signed XML has more than one signatures.
- OCSP based certificate validation is now supported.
- Unicode file names and certificate subject and issuer names are now supported.

Following object methods have been added:

- None

Following object properties have been added:

- CertValidationTransactionId
- IgnoreIncompleteSignature
- SignatureIndexToVerify
- UseOcsp
- OcspTrustedRespSignerCertPath
- OcspReqSignerPFXCertPath
- OcspReqSignerPFXCertPassword
- OcspResponderURL
- OcspTextResponse
- OcspB64Response
- CertRevocationDate
- IncludeOcspResponse

Bug Fixes

- None.

Other Changes

- None.

Special Note for JDK 1.3 users:

If you are using JDK 1.3 or earlier, you would need to copy the Jacob.dll to the Windows\System32 or WINNT\System32 directory in order to access SecureXML object from your Java programs. JDK 1.4 and higher is able to find Jacob.dll in the PATH variable's directory list and hence Jacob.dll can be any where as long as it is pointed to by the PATH environment variable.

Version 2.1.137.38

New Features

- Split siging is now supported where the digest calculation is done on a server and final signature value calculation is done on client machine. It is most useful when signing large files.
- Fetched CRL data can now be included in the signed documents allowing offline certificate validation without fetching the CRL from the CA.
- Now SecureXML provides API for buffer based access to the signed reference/object data hence allow a file I/O free operation.

Following object methods have been added:

- SetActivePFXB64Data
- GetErrorDetail
- GetSignedInfoDigest
- GetX509CertificateChain
- SignSignedInfoDigest
- GetSignedInfoDigestFromByteArray
- ApplySignatureValue
- ApplySignatureValueGetByteArray
- GetSignedDocumentB64Str
- GetSignedDocumentByteArray

Following object properties have been added:

- IncludeCRLInSignature
- UsedCRLList
- PhysicalSignatureB64Str
- SignerCertificateChain
- SignatureImageId
- LicensedUserCount

Bug Fixes

- Passing of arrays of strings from JavaScript/Jscript has now been fixed. Earlier versions worked only with VBScript when it came to array input as in CRLLocation etc. object properties.

Other Changes

- PhysicalSignatureUsage has a new mode = 3, which makes SecureXML include the signature image passed via the PhysicalSignatureB64Str object property to the current signature being created.

Version 2.1.136.37

New Features

- Byte array based signature creation and verification methods have been added
- Additional base64 encoding/decoding methods have been added
- The runtime performance of SecureXML is now four times faster than previous versions.

Following object methods have been added:

- SignXMLByteArray
- SaveXMLByteArray
- ReadAllByteArray
- VerifyXMLByteArray
- SecureXMLVerifyByteArray
- SignXMLXpathByteArray
- SignXMLEnvelopedByteArray
- Base64DecodeByteArrayToFile
- Base64DecodeByteArrayToByteArray
- Base64EncodeByteArrayToByteArray
- Base64EncodeByteArrayToFile
- Base64EncodeStrToFile
- Base64EncodeStrToStr

Following object properties have been added:

- SignedDocumentCount
- LogLevel

Bug Fixes

- A problem related to UTF-16 encoded XML processing has been fixed.
- A problem related to multi-threaded server side operation has been fixed.

Other Changes

- CaptureOnce object property has been removed.

Version 2.1.135.36

New Features

- Now you can specify canonicalization method for all non-template based signatures (in template based signatures you can set the canonicalization method in your signature template).
- SecureXML now allows a user defined timeout for all cached CRLs. CRLs are refreshed at either the user defined timeout or the nextupdate time whichever comes first.
- Support for Netscape certificate store has been added. The minimum supported version of Netscape is Version 7.1
- Support for PEM formatted certificates has been added

Following object properties have been added:

- CRLCacheTimeoutInMinutes
- CanonicalizationMethod
- NetscapeStorePassword
- SecureXMLPath

Following object methods have been added:

- SetActivePEMFileCert

Bug Fixes

- Signature verification failed in rare cases.

Other Changes

- SetStoreName method now accepts “Netscape” as store name for accessing the Netscape certificate store.

Version 2.1.134.35

New Features

- You can now restrict the certificates available for use by specifying the certificate issuer name. The end users will only see certificates issued by the specified certificate issuer(s) when they sign any document.
- CRL Caching is now supported by SecureXML. By default it stores cached CRLs in the CRLCache table of the file C:\Program Files\Infomosaic\SecureXML\SecureXML.mdb. One can specify a database connection string to change the database as long as there is an ADO provider present for the database.

Following object properties have been added:

- AllowedCertIssuerNames
- CRLCacheDbConnectionString
- UseCRLCache

Bug Fixes

- None.

Other Changes

- When CRL Checking is not used, CRLValidationMethod property is not added to the signature being produced.

Version 2.1.133.34

New Features

- None

Bug Fixes

- ViewCertificate and GetX509Certificate methods did not handle plain hex format for certificate serial number correctly.

Other Changes

- CamValidationResponse now returns base64 encoded complete response data which includes the binary value of the CA Signed Message as well as the HTTP header preceeding that. It is the same behavior as it was before version 2.1.133.33

Version 2.1.133.33

New Features

- Now you can get signature properties by querying them by name after verifying a signature. If you already know the name of a property, you do not need to iterate through all the signature properties in order to access its value.
- Base64 decode functionality has been added.
- You can now gunzip files using SecureXML. It is useful when working with files signed using SecureSign Desktop Document Signer, which produces gzipped XML files (.sig files).
- Plain hex format for certificate serial number is now supported making it easy to read it and compare. The base64 encoded binary is still the default format. For using plain hex format, just set CertSerialNumberFormat to 1.
- You can now delete signature from a signed XML by giving its signature Id.

Following object properties have been added:

- CertSerialNumberFormat

Following object methods have been added:

- GetSigPropValueByName
- Base64DecodeBufferToFile
- Base64DecodeFileToFile
- GunZipFile
- SecureXMLVerifyFileToBuffer
- SecureXMLVerifyFileToFile
- DeleteSignatureFromXMLStr
- DeleteSignatureFromFile

Bug Fixes

- When verifying a tampered signed XML with SecureXMLVerify method, the method reported incorrect signer certificate information.

Other Changes

- GetCertificateInfo now also provides information about currently active certificate.
- CamValidationResponse now only returns the base64 encoded binary value of the CA Signed Message. Earlier it also included the HTTP header.

Version 2.1.132.32

New Features

- OCSP via Certificate Arbitrator Module (CAM) Server is now supported. The Certificate Arbitrator Module (CAM) is an application-level router that efficiently and consistently routes certificates from relying party programs to the issuing certificate authorities (CAs) for validation. By interfacing directly with the CAM, a relying party application will be able to interact seamlessly with multiple CAs.
- A new feature allows users to base64 encode their signed XML data before calling signature verification and instructs SecureXML to base64 encode signed XML before returning it to the caller.
- SecureXML nows implements config file based initialization for most object properties.

Following object properties have been added:

- UseCam
- CamServerHost
- CamServerPort
- AgencyId
- Base64EncodeXML
- Base64DecodeXML
- IncludeCamResponse
- CamValidationResponse

Bug Fixes

- CRL checking for a self issued certificate led to a memory access error. This bug should not affect any real world usage of SecureXML as any real PKI deployment has a root certificate other than the end certificate. This problem is relavant only in test scenarios and has been fixed.

Other Changes

- The SignatureAlgorithm object property has been removed.
- The GetXMLSignature object method has been removed.
- GetCertificateInfo now also provides information about PFX certificates.

Version 2.1.130.31

New Features

- None

Bug Fixes

- SecureXML incorrectly returned an empty authority constrained policy set under certain certificate policy settings. This problem was discovered while performing CAC tests for DoD Compliance and has now been fixed.

Other Changes

- Setting DoDCompliance = 1 now enables CertificateTrustExplicit and hence at least one trusted root certificate must be provided by setting TrustedRoots property.
- When signing a signature template, the signature element of the template must have a signature Id.

Version 2.1.129.31

New Features

- Encryption (for multiple recipients) and decryption of strings and files are now supported. In order to use the encryption features, a new license file is required.
- Additional certificate checking parameters such as Certificate Policy, Certificate Chain and Certificate Path Length have been added.
- SecureXML is now DoD JITC compliant. Applications can now easily become DoD JITC compliant by setting DoDCompliance property to 1. Please note that the end application will still need to be tested by the JITC but its certification is almost certain.
- Now CRL files and/or their URLs can be provided externally if the certificates don't contain this information.
- Certficate trust can now be limited to a set of selected root certificates.
- SecureXML now outputs the result of applying user contrained policies as well as authority contrained policies.
- You can now provide a set of detached object URIs when creating signature. SecureXML will add additional references for these detached objects and include them as part of the signature.

The following object methods have been added:

- EncryptStr
- DecryptStr
- EncryptFile
- DecryptFile
- VerifyActiveCertificate
- ReadAllBase64

Following object properties have been added:

- RecipientCertificates
- RecipientCertificateFiles
- RecipientCertificateStore
- DoDCompliance
- TrustedRoots
- CRLLocation
- CertificatePolicy
- AuthorityConstrainedPolicy
- UserConstrainedPolicy
- CertificatePolicyChecking
- CertificateChainValidation
- CertificatePathLengthChecking
- CertificatePolicyExplicit
- CertificateTrustExplicit
- DetachedObjects
- AttachedObjects
- DecryptionPFXCertFile
- DecryptUsingPFXFileCert
- DecryptionPFXPassword
- XpathNamespace

Bug Fixes

None

Other Changes

- URL reference is now allowed for all signature modes (enveloped, enveloping and detached) of the Sign method.
- The whole set of X509 certificates, which make up the certificate chain are now included for each signed document unless the ExcludeSignerCertificate property is set to 1.
- SignDataStr now base64 encodes the input string before creating the signature. A corresponding base64 transform has been added hence the signed object is still the original string. This is done to allow for non-XML characters in the input string.

Version 2.0.128.30

New Features

- Java (Sun JVM 1.4.1) is now fully supported.
- Easy way to co-sign is now supported.
- Signing multiple files at once is now supported.
- A new method to sign any arbitrary string has been added. The method prepares appropriate XML template and creates the XML signature. The input string is preserved in the clear so the data signed can be seen by human eye. Also, co-signing the signed XML thus produced is supported.
- Setting SignatureID before invoking any of the signature creation methods is no longer required. SecureXML internally generated a GUID and assigns it as the SignatureID if it is not set already.
- Support for configuration file has been added. It allows users to initialize the object properties using data contained in an XML file. Additional method has been provided to set object properties using any arbitrary config file at run time.
- Now the users have an option not to include a copy of the signer's certificate for each signature being created. The default behavior is to include the certificate.
- Certificate selection window is now also supported on Windows 98 platform as well.

The following object methods have been added:

- SignDataStr
- CoSignXMLStr
- SignFiles
- CoSignFile

Following object properties have been added:

- ExcludeSignerCertificate
- ConfigFileName
- OverwriteFile

Bug Fixes

None

Other Changes

Now SecureXML Digital Signature release includes programming examples for the following languages:

- C
- C++
- Java
- Visual Basic
- Visual Basic Script
- JavaScript
- .Net family of languages
    - C#
    - VB.NET
    - ASP.NET using C#

Version 2.0.128.29

New Features

- HMAC or password (non PKI) based signing is now supported.
- Additional helper function ChangeOrAddProperty has been added to make working with signature properties easier.
- Support for user provided XPath expressions has been added via SignXMLXpathStr method.
- Support for exported base64 encoded certificates, not necessarily installed in the local machine store, has been added. It allows users to store their certificates along with their public key in a central store and keep their private keys in either smart cards or USB tokens and hence be mobile within their enterprises and be able to sign as long as they can access then public keys stored in the central storage.
- A new property DetailedVerificationFlag has been added which when set to zero (the default is 1), prevents Verify() and VerifyXMLStr() methods from extracting detailed verification information from the signed XML. It makes the signature verification faster. If the signer details and the signed document details are needed, set the DetailedVerificationFlag back to 1 and call Verify() / VerifyXMLStr() again. Now you have access to all the properties related to the signature just verified.
- A new method CaptureLiveSignature() has been added. It lets users capture live signature from signature pads and mouse without having to create XML Siganture. It is useful for applications where all you need is an image and a PKI based XML Signature is not required.

The following object methods have been added:

- SignXMLXpathStr
- ChangeOrAddProperty
- CaptureLiveSignature

Following object properties have been added:

- UseHMAC
- HMACPassword
- SignerCertificate
- DetailedVerificationFlag

Bug Fixes

1. A bug related to template signing has been fixed. The previous release failed to detect the presence of the signature template if it was nested several levels deep inside the XML node tree.
2. A bug related to CRL checking when CRL distribution point was provided as ldap:// has been fixed. Now SecureXML correctly verifies CRL status for certificates with both http:// and ldap:// type CRL distribution points.

Version 2.0.127.28

New Features

- Select certificate once and sign multiple documents and web pages without selecting certificate again.

The following object method has been added:

- SelectActiveCertificate

Bug Fixes

- SignXMLStr and SignXMLEnveloped functions reported error 16 or XML Parser Error when presented an XML containing native characters from Microsoft XMLDocument.xml via Internet Explorer. This happened due to change in encoding introduced by MS XML Parser to UTF-16. This version now handles it correctly.

Version 2.0.125.27

Other Changes

- The license check now allows signature verification on all client machines with or without a valid license file. On Windows 2000 Server and .NET Server verification still requires a valid license.

Bug Fixes

- IE crashed when switching to a new page after a call to GetError() function.

Version 2.0.125.26

New Features

1. Support for physical signature creation and addition to Enveloped Signatures has been added. There are three options provided for physical signature source: File, Mouse and Wintab (www.pointing.com) compatible Signature Pad.
2. Support for capture and additional of the foreground window image to enveloped signature has been added.

Following Object Properties have been added:

- PhysicalSignatureUsage
- CaptureOnce
- PhysicalSignatureFile
- CapturedSignatureFile
- AddWindowImage

Bug Fixes

None

Version 1.9.125.25

Bug Fixes

1. Support for Windows 98SE added again. The release 1.9.124.24 used a function from platform SDK which is not supported on Windows 98SE. This function has been replaced by a backwards compatible function.

New Features

- Support for certificate validity against certificate revocation list added.

Following Object Properties have been added:

- CrlChecking
- SigCertStatus

Following Object Methods have been added:

- VerifyX509CertCRL
- VerifyPFXCertCRL

Version 1.9.124.24

Bug Fixes

1. Enveloping and Detached signature creation crashed the component when using Sign() method. Enveloped signature worked file. The problem was related to certificate handling and has been fixed.

Other Changes

License check now checks for Host Name and OS type as well.

Version 1.9.124.23

Bug Fixes

1. PFX file import failed for a certificate exported using PFXExportCertificate method with null password.
2. IE crashed when the user clicked the cancel button on the certificate selection window during signature creation process.
3. For some RSA public keys with large exponent values, the exponent value shown in the signature file was incorrect. This led to subsequent signature verification failures when using the public key for verification. The created signature was valid and one could verify it using the X509 certificate fine.

New Features

- Added support for automatic time stamping during signature creation process. Users can now either use the default NIST time server or specify their own time server. If time stamping is enabled, a new signature property is added to indicate the time stamp server used.

Following Object Properties have been added:

- TimeStampURL
- TimeStamping

SecureXML Digital Signature & Encryption User's Guide

Infomosaic SecureXML Digital Signature component makes adding support for XML Digital Signature in any web or standalone application very easy. Its simple to use interface are very intuitive. The majority of the XML processing is done in C making it highly efficient for high volume transaction applications. The following sections describe how you can use SecureXML from any of the supported languages: C, C++, Java, VB, VBScript, JavaScript, C# and VB.NET.

Configuring SecureXML via SecureXML.config File

When a SecureXML object is created, it looks for SecureXML.config file in the same directory where XMLSign.dll or SignatureL.dll (when using Java) is located. If such a file is found, it is parsed and its content is used to assign the initial values for various SecureXML object properties. The following shows the format and the properties that can be set using SecureXML.config file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
	<appSettings>
		<add key="AddWindowImage" value="0" />
		<add key="CrlChecking" value="1" />
		<add key="EnvelopingFlag" value="2" />
		<add key="PhysicalSignatureFile" value="" />
		<add key="PhysicalSignatureUsage" value="2" />
		<add key="TimeStampURL" value="http://time-b.timefreq.bldrdoc.gov:13" />
		<add key="TimeStamping" value="0" />
		<add key="CertificatePathLengthChecking" value="0" />
		<add key="CertificateChainValidation" value="0" />
		<add key="CertificatePolicyChecking" value="0" />
		<add key="CertificateTrustExplicit" value="1" />
		<add key="TrustedRootCount" value="2" />
		<add key="TrustedRoot" value="C:\DoDPKIClass3RootCA.cer" />
		<add key="TrustedRoot" value="C:\JITCDoDPKIClass3RootCA.cer" />
		<add key="CertificatePolicyExplicit" value="0" />
		<add key="CertificatePolicyCount" value="2" />
		<add key="CertificatePolicy" value="1.0.3.4.5.6" />
		<add key="CertificatePolicy" value="1.0.3.4.5.6.7.8" />
		<add key="CRLLocationCount" value="2" />
		<add key="CRLLocation" value="C:\temp\MyCrl_0.crl" />
		<add key="CRLLocation" value="C:\temp\MyCrl_1.crl" />
		<add key="OverwriteFile" value="1" />
		<add key="ExcludeSignerCertificate" value="0" />
		<add key="DetailedVerificationFlag" value="1" />
		<add key="Base64EncodeXML" value="0" />
		<add key="Base64DecodeXML" value="0" />
		<add key="CamServerHost" value="testcam.digsigtrust.com" />
		<add key="CamServerPort" value="7777" />
		<add key="UseCam" value="0" />
		<add key="AgencyId" value="Infomosaic SecureXML" />
		<add key="DoDCompliance" value="1" />
		<add key="AllowedCertIssuerCount" value="2" />
		<add key="AllowedCertIssuerName" value="C3 ID CA" />
		<add key="AllowedCertIssuerName" value="Infomosaic " />
		<add key="CRLCacheDbConnectionString" value="DRIVER={sql
server};SERVER=(local);DATABASE=SECUREXML;UID=sa;PWD=password" />
		<add key="UseCRLCache" value="1" />
		<add key="CRLCacheTimeoutInMinutes" value="120" />
	</appSettings>
</configuration>
```

Using SecureXML from Microsoft Visual Basic Script

The process of using SecureXML from VBScript either on the client side or on an ASP page on the server side starts with first creating an instance of the SecureXML Signature object by doing a CreateObject on the object name:

Set SigObj = CreateObject(“XMLSign.Signature”) on the client side and
Set SigObj = Server.CreateObject(“XMLSign.Signature”) on an ASP page for server side usage.

Access to all methods and properties are straight forward as their names remain exactly the same as in the XMLSign.idl file.

Example 1: PKI Signature, Co-Signing

```
‘Create SecureXML Signature object
Set SigObj = CreateObject("XMLSign.Signature")

‘Set a signature property
SigObj.ChangeOrAddProperty "Signer Name", "John Doe"

‘ Set ExcludeSignerCertificate to 1 if you don’t want to include a copy of the signer’s certificate in the signed XML
SigObj.ExcludeSignerCertificate = 1

‘ Just sign a simple data string
res = SigObj.SignDataStr ("This is my test data")

‘Co-sign the signed xml produced above
res = SigObj.CoSignXMLStr (res)

‘Check for errors
MsgBox(SigObj.GetLastError)

‘Save the signed xml containing two signatures to a file
SigObj.SaveXMLStr res, "TestFiles\c1.xml"

‘Verify the signed XML created above
res = SigObj.SecureXMLVerify(res)

‘Display verification result
MsgBox(res)
```

Example 2: HMAC or Password based signatures

```
‘Create SecureXML Signature object
Set SigObj = CreateObject("XMLSign.Signature")

‘Set Signature Id
SigObj.SignatureID(0) = "MySignature"

‘ Set EnvelopingFlag to Enveloped mode
SigObj.EnvelopingFlag = 2
‘ Enable HMAC signing
SigObj.UseHMAC = 1
‘ Set the signing password
SigObj.HMACPassword = "password"

‘Set some other signature property
SigObj.ChangeOrAddProperty "Location", "Honolulu"

‘Create the signature
res1 = SigObj.Sign("TestFiles\catalog.xml")
MsgBox res1 & "Last Error = " & SigObj.GetLastError
```

```
‘Enable SaveXMLSignature to overwrite existing files
SigObj.OverwriteFile  = 1

‘Save the signature created above
SigObj.SaveXMLSignature "TestFiles\c1.xml"
MsgBox "Signature Saved As TestFiles\c1.xml"
```

Using SecureXML from JavaScript on Windows

The process of using SecureXML from JavaScript either on the client side or on an ASP page on the server side starts with first creating an instance of the SecureXML Signature object by doing a new ActiveXObjecton on the client side and Server.CreateObject for server side usage:

var SigObj;
SigObj = new ActiveXObject(“XMLSign.Signature”) on the client side and
SigObj = Server.CreateObject(“XMLSign.Signature”) on an ASP page for server side usage.

Alternatively on the client side, the html may include an object tag with classid set to the classid of SecureXML Signature object in order to instantiate the object. This method works faster than trying to create an object by calling new ActiveXObject. The new ActiveXObject is available only in Internet Explorer.

If your application needs to be supported on Netscape/Mozilla browsers, you must declare an <object> tag in your html page in order to use SecureXML from JavaScript on that page.

<object id=SigObj classid=CLSID:D300C133-A6F6-4FB4-A734-4865FBF5A3B1> </object>

In addition to the object tag please make sure that the following line has been added to your activex.js file located in C:\Program Files\Netscape\Netscape\defaults\pref directory:

pref("capability.policy.default.ClassID.CIDD300C133-A6F6-4FB4-A734-4865FBF5A3B1", "AllAccess");

Access to all methods and properties are straight forward as their names remain exactly the same as in the XMLSign.idl file.

```
var signedXML=null;
var signedXMLFile=null;
var count;
var newImagePath;

function SignNow()
{
        var SigDate, ErrorCode;
        SigDate = new Date();
// Create an instance of the SecureXML Digital Signature Object
         var SigObj = new ActiveXObject("XMLSign.Signature");
// Set a few object properties
        SigObj.SignatureID (0) = "MySignature";
        SigObj.Properties (0,0) = "SignatureDate = " + SigDate.toString();
// Lets capture a live signature image
        SigObj.PhysicalSignatureUsage = 2;
// Select the certificate to be used for subsequent signing.
// You can skip this step if you would rather see a certificate selection window
        SigObj.SetActiveCertificate(SigObj.GetCertificateInfo(0, 1));
// Sign a simple data string
        signedXML = SigObj.SignDataStr(“This is my input”)
        ErrorCode = SigObj.GetLastError()
        if (ErrorCode != 0) {
                if (ErrorCode == 13) {
                        alert("No Certificate Selected");
                }
                else {
                  if (ErrorCode == 45) {
                     alert("You either do not have signature creation license or your license has expired");
                  }
                  else {
                   alert("Signature Creation Failed. Last Error = " + ErrorCode + " Error Stack = " + SigObj.GetError());
                  }
```

```
                    }
                    return false;
            }
            else { // Signature was created successfully
              var uriCount, docUri, uriIndex, imageURI;
// Verify the signature we just created
              SigObj.VerifyXMLStr(signedXML);
              signatureId = SigObj.SignatureID(0);
// The following code shows how to extract the signature image that was captured during signature creation
              uriCount = SigObj.TotalUriCount;
              imageURI = "#SignatureImage_" + signatureId;
              for (uriIndex = 0; uriIndex < uriCount; uriIndex++) {
                    docUri = SigObj.DocumentURI(0,uriIndex);
                    if (docUri == imageURI)  {
                      newImagePath = SigObj.SignedDocumentPath(0,uriIndex);
                     // newImagePath now contains the complete path for the file containing the signature image
                      break;
                    }
            }
        }
       return false;
}
```

Using SecureXML from Microsoft Visual Basic 6.0

There are two ways to create an instance of the SecureXML Digital Signature object in Visual Basic 6.0. One way is to follow the usage presented for Visual Basic Script. This method, also know as late binding method has a disadvantage that you don't get the benefit of the *Intellisense* built into the VB 6.0 development environment. Also there is a run-time penalty for late binding. The preferred way is to create a reference to the SecureXML Digital Signature type library in your VB 6.0 development environment and then declare a typed object. In order to create a reference to the SecureXML Digital Signature Type Library, please click on ***Project*** pull down menu and select ***References…*** At this point you will be presented with a dialog box with a list of available references. Please locate and select an entry called "Infomosaic SecureXML Digital Signature 1.0 Type Library". If you don't find this entry, click on the browse button and locate and select XMLSign.dll in C:\Program Files\Infomosaic\SecureXML\ directory or the SecureXML install directory if you chose a different directory during installation.

After you have created a reference to the SecureXML Digital Signature Type Library you would need to declare a variable of this new type. This is done as follows:

```
Public SigObj As XMLSIGNLib.Signature
```

Later in your code, you would need to create an instance of the SecureXML object as follows:

```
Set SigObj = New XMLSIGNLib.Signature
```

Now SigObj points to an instance of the SecureXML Digital Signature object and the VB 6.0 development environment can help you use it by presenting you with a list of methods and properties available for this object. For the most part if you go through the sample VB 6.0 project included with the SecureXML download, you will have your first VB application running within one or two days.

With VB, all the method and property names remain the same as in the IDL file and the VB 6.0 *Intellisense* can help you use them without having to read this guide in great detail.

Using SecureXML from Microsoft Visual C 6.0

Before compiling any C programs, please make sure that XMLSign.idl file is included in your project and it is part of the build process. You may have to manually specify the build command for the idl file if one is not added automatically by Visual Studio. You can do this by specifying ***midl $(InputPath)*** for the custom build command and an output directory for XMLSign.idl file by right clicking XMLSign.idl in the ***FileView*** and choosing ***Settings***.

```
/* Defining COBJMACROS allows you to call the MIDL generated wrappers for SecureXML methods and properties.
 * It makes using SecureXML from C a little easier.
*/
#define COBJMACROS

#include <stdio.h>
/* The following two files are generated by Visual Studio provided IDL compiler: MIDL */
#include "XMLSign.h"
#include "XMLSign_i.c"

void main()
{
        HRESULT hr;
        IUnknown *pUnknown;
        ISignature *pSig;
        BSTR tmpFileName;
        BSTR inputData;
        LONG status;
        int signatureCount;
        int certificateCount;
        int referenceCount;
        BSTR signerName;
        BSTR certIssuer;
        BSTR docURI;
        BSTR sigId;
        int i,j;

        /* Initialize Windows COM Infrastructure */
        hr = CoInitialize(NULL);
        if (FAILED(hr))     {
                printf("%s","CoInitialize failed.\n");
                exit(0);
        }

        /* Create an instance of SecureXML Digital Signature */
        hr = CoCreateInstance(&CLSID_Signature, NULL, CLSCTX_INPROC_SERVER,
                                                        &IID_IUnknown, (void **) &pUnknown);
        if (FAILED(hr))     {
                printf("%s","CoCreateInstance failed.\n");
                exit(0);
        }

        /* Create a XMLSign.Signature Object and obtain it's handle */
        hr = ISignature_QueryInterface(pUnknown, &IID_ISignature, (void **) &pSig);
        if (FAILED(hr))     {
                printf("%s","QueryInterface failed.\n");
                exit(0);
        }

        /* Now pSig is a pointer to a XMLSign.Signature Object */
        /* Since we don't want any more Signature objects, release the handle to SecureXML */
        ISignature_Release(pUnknown);
```

```
	/* Lets invoke some methods and set a few properties to make sure we can use it */
	hr = ISignature_ReadAll(pSig, L"TwoSignature.xml", &inputData);
	if (FAILED(hr))	{
		printf("%s","ReadAll failed.\n");
		exit(0);
	}

	/* Verify the signature from the above file */
	hr = ISignature_VerifyXMLStr(pSig, inputData, &status);
	if (status) {
		printf("Signature Verified Successfully\n");
		ISignature_get_SignatureCount(pSig, &signatureCount);
		printf("Signature Count = %d\n", signatureCount);
		for (i=0; i < signatureCount; i++) {
			ISignature_get_SignatureID(pSig, i, &sigId);
			ISignature_get_SignerSubject(pSig, sigId, &signerName);
			ISignature_get_CertIssuer(pSig, sigId, &certIssuer);
			printf("Signature #%d, Signer Name = %ws\n\n", i, signerName);
			printf("Signature #%d, Certificate Issued By %ws\n\n", i, certIssuer);
			ISignature_get_TotalUriCount(pSig, &referenceCount);
			for (j=0; j < referenceCount; j++) {
				ISignature_get_DocumentURI(pSig, i, j, &docURI);
				printf("Signature #%d Document URI #%d = %ws\n", i, j, docURI);
			}
		}
	}

	hr = ISignature_get_CertificateCount(pSig, &certificateCount);
	printf("You have %d PKI certificates installed in your computer\n", certificateCount);
	if (status == 0) {
		printf("Signature verification failed\n");
	}

	/* Lets create a signature */
	hr = ISignature_put_SignatureID(pSig, 0, L"MySignature");
	hr = ISignature_put_EnvelopingFlag(pSig, 2); /* Create Enveloped Signature */
	hr = ISignature_put_PhysicalSignatureUsage(pSig, 2); /* Lets capture a live signature image */
	hr = ISignature_Sign(pSig, L"catalog.xml", &tmpFileName); /* Sign the XML contained in catalog.xml file */

	hr = ISignature_put_OverwriteFile(pSig, 1); /* Force file overwrite */
	hr = ISignature_SaveXMLSignature(pSig, L"catalogSigned.xml");
	printf("Signature saved as catalogSigned.xml\n");

	/* Lets verify the signature we just created */
	hr = ISignature_Verify(pSig, L"catalogSigned.xml", &status);
	if (status)  {
		ISignature_get_SignatureCount(pSig, &status);
		printf("Signature Verified Successfully\n");
		printf("Signature Count = %d\n", status);
	}
	ISignature_Release(pSig); /* We are done */
	CoUninitialize();
}
```

Using SecureXML from Microsoft Visual C++ 6.0

Before compiling any C++ programs, please make sure that XMLSign.idl file is included in your project and it is part of the build process. You may have to manually specify the build command for the idl file if one is not added automatically by Visual Studio. You can do this by specifying ***midl $(InputPath)*** for the custom build command and an output directory for XMLSign.idl file by right clicking XMLSign.idl in the ***FileView*** and choosing ***Settings***.

```
#include "stdafx.h"
/* The following two files are generated by Visual Studio provided IDL compiler: MIDL */
#include "XMLSign.h"
#include "XMLSign_i.c"

int main(int argc, char* argv[])
{
        HRESULT hr;
        IUnknown *pUnknown;
        ISignature *pSig;
        BSTR verifyR;
        BSTR inputData;

        /* Initialize Windows COM Infrastructure */
        hr = CoInitialize(NULL);
        if (FAILED(hr))     {
                printf("%s","CoInitialize failed.\n");
                exit(0);
        }

        /* Create an instance of SecureXML Digital Signature */
        hr = CoCreateInstance(CLSID_Signature, NULL, CLSCTX_INPROC_SERVER,
                                                        IID_IUnknown, (void **) &pUnknown);
        if (FAILED(hr))     {
                printf("%s","CoCreateInstance failed.\n");
                exit(0);
        }

        /* Create a XMLSign.Signature Object and obtain it's handle */
        hr = pUnknown->QueryInterface(IID_ISignature, (void **) &pSig);
        if (FAILED(hr))     {
                printf("%s","QueryInterface failed.\n");
                exit(0);
        }

        /* Now pSig is a pointer to a XMLSign.Signature Object */
        /* Since we don't want any more Signature objects, release the handle to SecureXML */
        pUnknown->Release();

        /* Lets invoke some methods and set a few properties to make sure we can use it */
        hr = pSig->ReadAll(L"TwoSignature.xml", &inputData);
        printf("After ReadAll\n");
        if (hr != S_OK)      {
                printf("%s","ReadAll failed.\n");
                exit(0);
        }

        hr = pSig->SecureXMLVerify(inputData, &verifyR);
        printf("After SecureXMLVerify\n");
        if (hr != S_OK)      {
                printf("%s","SecureXMLVerify failed.\n");
                exit(0);
        }
```

```
	printf("%ws", verifyR);

	/* Now lets create a new signature */
	hr = pSig->put_SignatureID(0, L"MySignature");
	BSTR tmpSignedXMLFileName;
	hr = pSig->Sign(L"catalog.xml", &tmpSignedXMLFileName);
	hr = pSig->put_OverwriteFile(1);
	hr = pSig->SaveXMLSignature(L"catalogSigned.xml"); /* catalogSigned.xml now contains the signed xml */

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

Using SecureXML from Java

Java classes for SecureXML are included as securexml.jar file. Java with SecureXML requires SignatureL dynamic library file (on Windows it is SignatureL.dll, on Linux machines it is libSignatureL.so, on the upcoming Mac OS X release it is libSignatureL.dylib) also included with SecureXML SDK. securexml.jar file needs to be added to your CLASSPATH environment variable and the SignatureL.dll / libSignatureL.so / libSignatureL.dylib should be in your PATH environment variable. If you are using SecureXML from a custom Java applet, you would need to sign your applet with a code signing certificate issued by a known certificate authority.

The following code shows how to instantiate Signature object and invoke methods on it:

```
import infomosaic.securexml.*;

public class SignTempPlusDataStr
{
  public static void main(String[] args)
  {
try{
    Signature s1 = new Signature(); /* Creates an instance of the SecureXML Signature object */
    String outFilePath;
    System.out.println("Certificate Count="+ s1.getCertificateCount());   /* Lets report the number of certificates installed in the
                                                                              * local windows certificate store for the currently logged
                                                                              * on user  */
   String certId = s1.getCertificateInfo(0,1); /*Get the certificate Id of the first certificate in the Windows certificate store*/
   s1.setActiveCertificate(certId);              /*Make this the active certificate */

    String fileData = s1.readAll("signature.tmpl");                    /* Read the file to be signed */
    String outFileData = s1.signXMLStr(fileData, "MySignature");       /* Sign the XML read from the file */
    System.out.println("Signed XML= "+ outFileData);                   /* Show the signed XML */
    s1.saveXMLStr(outFileData, "signatureSigned.xml");                 /* Save the signed XML to a file */
    int result = s1.verify("signatureSigned.xml");                     /* Verify the signature we just created */
    System.out.println("Signature verification result= "+ result);
    String verifyResult = s1.secureXMLVerify(outFileData);
    System.out.println("secureXMLVerify = "+ verifyResult);
    s1.setSignatureID(0, "MySignature1234");                           /* Lets create another signature */
    s1.setExcludeSignerCertificate(1);
    s1.setProperties(0, 0, "Date=12/19/2002");
    s1.setPhysicalSignatureUsage(2);
    outFileData = s1.signDataStr("This is my input");                  /* This time, lets just sign a simple data string */
    System.out.println("Signed Data Str XML= "+ outFileData);
    s1.saveXMLStr(outFileData, "dataSigned.xml");
    result =  s1.verify("dataSigned.xml");                             /* Verify the signature we just created */
    String propertyStr = s1.getProperties(0,0);
    System.out.println("Signature Property 0 = "+ propertyStr);

/* The following code passes a list of files to be signed */
/* The output XML contains Base64 encoded file content along with the signature */
    String [] fileList = new String[3];
    fileList[0] = "f1.doc";
    fileList[1] = "f2.txt";
    fileList[2] = "f3.doc";
    String outFileName = s1.signFiles(fileList, "3FilesSigned.xml");
    System.out.println("Out File Name = " + outFileName);
    s1.destroy(); // Call this method to cleanup resources: it will delete temporary files and free memory
}
catch (Exception e){
}
  }
}
```

Using SecureXML Java Applet from Web Browsers

The SecureXML Java Applet is available both as a signed jar file and a signed cab file. Signed jar file is used when Sun JVM is used by the browser and the signed cab file is used when Microsoft JVM is used. The following html/JavaScript example shows how to include the Infomsaic SecureXML Java Applet in your web pages and how to use it:

```
<html>
      <body>
            <applet codebase = "."
                  archive  = "securexmlapplet.jar"
                  code     = "infomosaic.securexml.applet.XMLSignApplet"
                  id="securexmlApplet"
                  name="securexmlApplet"
                  codebase="./"
                  scriptable="true"
                  width    = "0"
                  height   = "0"
                  hspace   = "0"
                  vspace   = "0"
                  align    = "middle"
                  MAYSCRIPT VIEWASTEXT>
            <PARAM name="cabbase" value="securexmlapplet.cab" />
            </applet>
<Script language="JavaScript">
var newLink;
var linksEnabled=false;
var sig;
function disableLinks(){
   linksEnabled=false;
   var aLnk=document.getElementsByName("buttons");
   for (i=0; i<aLnk.length; i++) {
     aLnk[i].disabled = true;
   }
}

function enableLinks(){
   linksEnabled=true;
   var aLnk=document.getElementsByName("buttons");
   for (i=0; i<aLnk.length; i++) {
     aLnk[i].disabled = false;
   }
}
var outStr;
function getAllInfoFromCac(smCard)
{
      outStr = new String();
      outStr += "First Name = " + smCard.getFirstName() + " \n";
      outStr += "Blood Type Code = " + smCard.getBloodTypeCode() + " \n";
      outStr += "Branch of Service Code = " + smCard.getBranchOfServiceCode()
+ " \n";
      outStr += "Civilian Healthcare Entititlement Code = " +
smCard.getCHCCode() + " \n";
      outStr += "Civilian Healthcare Entititlement End Date = " +
smCard.getCHCEndDate() + " \n";
      outStr += "Cadency = " + smCard.getCadency() + " \n";
      outStr += "Card Expiry Date = " + smCard.getCardExpiryDate() + " \n";
```

```
      outStr += "Card Issue Date = " + smCard.getCardIssueDate() + " \n";
      outStr += "Commissary code = " + smCard.getCommissaryCode() + " \n";
      outStr += "Date Demographic Data on Chip Expires = " +
smCard.getDateCacDataExpires() + " \n";
      outStr += "Date Demographic Data was loaded on Chip = " +
smCard.getDateCacDataLoaded() + " \n";
      outStr += "Date of birth = " + smCard.getDateOfBirth() + " \n";
      outStr += "Direct care code = " + smCard.getDirectCareCode() + " \n";
      outStr += "Direct care end date = " + smCard.getDirectCareEndDate() + "
\n";
      outStr += "DoD Contractor Function Code = " +
smCard.getDoDContractorFunctionCode() + " \n";
      outStr += "Exchange Code = " + smCard.getExchangeCode() + " \n";
      outStr += "Gender Code = " + smCard.getGenderCode() + " \n";
      outStr += "Last Name = " + smCard.getLastName() + " \n";
      outStr += "MWR Code = " + smCard.getMWRCode() + " \n";
      outStr += "Meal plan Code = " + smCard.getMealPlanCode() + " \n";
      outStr += "Middle Name = " + smCard.getMiddleName() + " \n";
      outStr += "Non Medical Benefits Association End Date = " +
smCard.getNonMedicalBenefitsEndDate() + " \n";
      outStr += "Non US Government Agency Code = " +
smCard.getNonUSGovernmentAgencyCode() + " \n";
      outStr += "Organ Donor Code = " + smCard.getOrganDonorCode() + " \n";
      outStr += "Pay Grade Code = " + smCard.getPayGrade() + " \n";
      outStr += "Pay Plan Code = " + smCard.getPayPlan() + " \n";
      outStr += "DoD EDI Person Identifier = " + smCard.getPersonDoDEDI() + "
\n";
      outStr += "Person Identifier = " + smCard.getPersonIdentifier() + "
\n";
      outStr += "Person Identifier Type Code = " +
smCard.getPersonIdentifierTypeCode() + " \n";
      outStr += "Personnel Category Code = " +
smCard.getPersonnelCategoryCode() + " \n";
      outStr += "Personnel Entitlement Condition Code = " +
smCard.getPersonnelEntitlementConditionTypeCode() + " \n";
      outStr += "Rank Code = " + smCard.getRank() + " \n";
      outStr += "US Government Agency Code = " +
smCard.getUSGovernmentAgencyCode() + " \n";
outStr += "Certificate #0 CN = " + smCard.getCN(0) + " Email Address = " +
smCard.getEmailAddress(0) + " \n"; //Get CN and Email Address from location
0100
outStr += "Certificate #1 CN = " + smCard.getCN(1) + " Email Address = " +
smCard.getEmailAddress(1) + " \n"; //Get CN and Email Address from location
0101
outStr += "Certificate #2 CN = " + smCard.getCN(2) + " Email Address = " +
smCard.getEmailAddress(2) + " \n"; //Get CN and Email Address from location
0102
      return outStr;
}
function invokeSign()
{
      var certCount=0;
      var certId = new String();
        var ainfo = document.securexmlApplet.getAppletInfo();
        sig = document.securexmlApplet.getSignature();
        cac = document.securexmlApplet.getSmartcard();
      try
```

```
      {
            cac.connectToCard("Some Card"); // “Some Card” input parameter is
ignored.
            // logonToCard is optional step. If the user has already
authenticated to the CAC then it is not required.
            cac.logonToCard(passwordForm.cacPassword.value);
            CacOutputArea.innerText = getAllInfoFromCac(cac);

            sig.selectActiveCertificate();
            signatureOutputArea.innerText = sig.signDataStr("This is a test")
+ sig.getError();
      }
      catch(Exception)
      {
            alert(ainfo + ": Caught exception " + Exception.toString());
            return false;
      }
      return true;
}
</Script>
            <form name="passwordForm">
            <b>Please enter Cac password:</b>
            <input type=text name="cacPassword" maxlength=50 id="cacPassword"
size="50">
            <input type="button" name="buttons" value="Click Me"
onclick="return invokeSign()">
            </form>
            <div id="CacOutputArea"></div>
            <div id="signatureOutputArea"></div>
      </body>
</html>
```

Using SecureXML from Microsoft .NET

In order to use SecureXML Digital Signature from the .NET development environment, you would need to add a reference to this component in your project environment. In order to create a reference to the SecureXML Digital Signature Type Library, please click on Project pull down menu and select Add Reference… At this point you will be presented with a dialog box with a list of available references. Click on "Com" tab and then locate and select an entry called "Infomosaic SecureXML Digital Signature 1.0 Type Library". If you don't find this entry, click on the browse button and locate and select XMLSign.dll in C:\Program Files\Infomosaic\SecureXML directory or the SecureXML install directory if you chose a different directory during installation.

Once a reference has been created please do the following:

From C#:

```
public XMLSIGNLib.Signature SigObj;
SigObj = new XMLSIGNLib.SignatureClass();
if (SigObj.CertificateCount == 0)
{
        MessageBox.Show("You don't have any PKI Certificates installed. Only HMAC Signatures can be created",
                "Warning", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
```

From VB.NET:

```
Public SigObj As XMLSIGNLib.Signature
SigObj = New XMLSIGNLib.Signature()
If SigObj.CertificateCount = 0 Then
        MessageBox.Show("You don't have any PKI Certificates installed. Only HMAC Signatures can be created",
                "Warning", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
End If
```

Using SecureXML on Linux and Mac OS X (10.3 Panther & 10.4 Tiger)

Overview

SeccureXML on Unix platforms has only two interfaces, Java Applet interface and a Java Class Interface. Both these interfaces use the same shared library file (libSignatureL.so on Linux and libSignatureL.jnilib on Mac). Java class uses securexml.jar file in addition to the shared library file. SecureXML implementation on Unix platforms uses Mozilla crypto API for all low level operations and hence mozilla shared libraries must be present during run time for SecureXML's operations. It uses libxml2 for all XML processing and hence libxml2 must also be installed. If OCSP based certificate validation is needed, OpenSSL must also be present.

SecureXML has been tested with the following versions of above required components:

OpenSSL	:	0.9.6b
Libxml2	:	2.6.5
Mozilla	:	1.7 (Firefox 1.0)

Server Side Operations Using Java

There are no differences between SecureXML usages from Java as compared with those on Windows. If you are using Apache Tomcat application server for hosting your server side beans/servlets or using JSP, you would need to put securexml.jar file in the webapps/yourApp/WEB-INF/lib folder and libSignatureL.so in a folder pointed to by the path environment variable or it can be put in the same folder where you application's .class files reside (normally in the webapps/yourApp/WEB-INF/classes/yourApp) folder.

Using Certificates

In order to use Firefox certificate store for signer certificate selection, two things need to be done:

1. Set the Certificate Store password (if the store is password protected) by setting the NetscapeStorePassword object property to the right value, and
2. Invoke SetStoreName method with "Netscape" (case sensitive) as the input parameter.

Since, Firefox certificate store is the only certificate store supported on Linux and Mac platforms, the above two steps must be performed before attempting to perform any operations involving certificates including signature creation.

Browser Based Operations Using JavaScript

Please refer to the description of the SecureXML Java Applet usage presented earlier in this guide.

Creating XML Digital Signature

There are several ways to sign digital content to produce an XML Signature. SecureXML Digital Signature provides the following methods to accomplish this:

1. Sign
2. SignXMLStr / SignXMLByteArray
3. SignXMLEnveloped / SignXMLEnvelopedByteArray
4. SignFile
5. SignHTML
6. SignXMLXpathStr / SignXMLXpathByteArray
7. SignDataStr
8. CoSignXMLStr / CoSignXMLByteArray
9. SignFiles
10. CoSignFile

Depending on how you want the input to be interpreted and how you want the output to be produced one or the other may be most suitable to your application.

If you start with an XML file and would like to see your XML plus the signature in the output signed XML file you can use either Sign, SignXMLStr or SignXMLEnveloped methods. If you start with any arbitrary file containing any digital content, you can use Sign, SignFile or SignFiles methods.

Using Sign

Sign method can create all three types of XML Signatures viz. Enveloping, Enveloped and Detached. In order to use this method, one needs to do some initial setup as follows:

1. Set Signature ID (Optional Step)
2. Set Enveloping Flag (Enveloping flag = 0, 1 or 2 for detached, enveloping or enveloped signatures respectively)
3. Call SetActiveCertificate to assign the signer certificate. A certificate selector window will pop up if an active certificate is not set.
4. Set Properties if needed.
5. Call Sign with the file or URL as the input.

Here is a short VB Script listing for using Sign method

```
Dim res, CertificateCount

Set SigObj = CreateObject("XMLSign.Signature")
CertificateCount = SigObj.CertificateCount

if CertificateCount = 0 Then
        MsgBox("No PKI Certificates Found")
Else
        ' Setting SignatureID is required
        'The index passed to SignatureID is ignored.
        SigObj.SignatureID(0) = "MySignature"

        ' Enveloping flag = 0 for detached signature (default)
        ' Enveloping flag = 1 for enveloping signature
        ' Enveloping flag = 2 for enveloped signature, the SignatureID must
        ' match the signature id used in the template file
        SigObj.EnvelopingFlag = 2

        'Set Properties (optional)
        ' The first parameter is ignored. The second parameter can vary between 0 and 31
        ' Internally the PropertyCount is set to the last index used
```

```
        ‘ So if you set Property #1 first and then #0, the signature will have only one property (the one ‘corresponding to #0)

        SigObj.Properties(0,0) = "SignatureDate = " & Date
        SigObj.Properties(0,1) = "SignatureTime = " & Time

        ‘ Now create the signature file. SecureXML stored the output in a temp file and returns its full path
        res = SigObj.Sign("c:\temp\input.xml")

        ‘ Check for errors
        MsgBox(res & SigObj.GetLastError)

        ‘save the temp file to a known location
        SigObj.SaveXMLSignature("c:\temp\inputSigned.xml")
        Set SigObj = Nothing
        MsgBox("Done")
End If
```

Effect of EnvelopingFlag on signature creation:

Case 1: EnvelopingFlag = 0 (default), creates a detached signature

The input file/URL is treated as binary octet stream and its digest value is calculated and signed. The signed info section contains a URI reference to the input file/URL. The data just signed is not contained in the signature XML.

Case 2: EnvelopingFlag = 1, creates an enveloping signature

A new XML signature envelope is created and a base64 encoded version of the data contained in the input file/URL is added as an object reference for the signature. If the input file was an XML, the data is still in the output signed XML but you can’t see it since it has been base64 encoded. You can of course retrieve it by base64 decode process.

Case 3: EnvelopingFlag = 2, creates an enveloped signature

The input XML is parsed and an empty <Signature> element with Id=SignatureID is searched. If found, the element pointed to by its signed info section is processed and the resulting digest and signature values are populated. If the input XML has no empty signature elements, a new signature element is created, the signed info now has Enveloped Signature Transform by default. The reference URI is set to “” which when used with the enveloped signature transform, implies that everything in the current XML, expect for the signature element, is signed. So now no additional signatures can be added to this output XML in an enveloped fashion since it will break the signature.

Using SignXMLStr

This allows for a XML signature template to be used for producing a signature. The input is an XML string and not a file name or URL. Other than that, it behaves just like Sign method with EnvelopingFlag = 2.

Using SignXMLEnveloped

This is a special case of SignXMLStr where entire input XML is always signed using the enveloped signature transform. One can set SignatureID and Properties for this new signature being added. The function parses the input XML string but doesn’t look for a SignatureID. It simply adds a fresh <Signature> element. This function is useful in Web Form signing situations where you have the XML as a string (not in a file, so you can’t use Sign), all you want to do is sign the whole thing and you know that you don’t need additional signatures to be added to this XML. The resulting signed XML can be directly sent to a database using appropriate SQL statements.

Using SignFile

This function allows for multiple people to sign an arbitrary input file. The Signed Info element is created to point to base64 encoded file content. One can fetch the original file content by calling GetSignedFileObject method on the output signature created using SignFile method.

Both input parameters refer to local files. If the output file already exists, it is overwritten.

Using SignHTML

This function allows for signing entire html page currently under display. If the page contains any forms which the user has filled out, the filled values are included in the signed content. If the page contains any images, they are not included in the signed content. In other words what you sign is what you see when you do a "View Source" on the current page with a difference that the current state of the form values and selections is also included.

The input parameter is "window.document" and the output is the signed XML string.

Using SignXMLXpathStr

This function is similar to SignXMLStr except that it adds one additional parameter for providing Xpath expressions. This function will save you the trouble of preparing a signature template complete with such details such as canonicalization methods and transforms. All you need to provide is the parent XML you want to sign a component of, the Xpath expression which will give the node sets you want signed and the signature Id you want to assign to this signature. Please note that if a signature element with matching Id is found, the Xpath expression is ignored and this function behaves exactly as SignXMLStr.

Using SignDataStr

The SignDataStr method provides a simple way to sign a non-XML text data. You basically start with a character string in a buffer and invoke SignDataStr. What you get back is a string which contains an XML which has the original string as the content of a <SignedObject> XML element along with the digital signature. The input string is base64 encoded to allow for characters not allowed in XML.

Using SignFiles

The SignFiles method allows you to provide a list of files to be signed. The output XML contains the content of each of the input files base64 encoded under individual <SignedObject> XML elements. The <Signature> element contains references to each of these <SignedObject> elements.

Using CoSignFile

If you used SignDataStr, SignFile or SignFiles to create your XML Signature, you can invoke CoSignFile if the signed output is contained in a file. This method as the name suggests, creates a new signature with the same <SignedObject> references as the previous signature contained in the input file. The input file is first subjected to signature verification and is processed only if the verification was successful.

Using CoSignXMLStr

This method is identical to the CoSignFile method except the input and the output are buffer based instead of file. In fact CoSignFile uses CoSignXMLStr internally after it has read the file content.

Split Signing in a Web Application

SecureXML supports client-server collaborative signature creation or split signing, which could streamline and optimize computing resource utilization and allow for better control on application settings during signature creation process. The following is a brief overview of how this process is supposed to work.

Start on the client side:

```
Set clientSigObj = CreateObject("XMLSign.Signature") ‘Create SecureXML object
‘The following steps related to certificate selection, certificate chain access etc. on the client side are required if the signed XML
‘ must have certificate validation information included. Otherwise certificate selection can be deferred until after digest
‘ calculation has been completed.
certId = clientSigObj.SelectActiveCertificate ‘Shows a dialog box with certificate selection window. Save certId to be used later
certChain = clientSigObj.GetX509CertificateChain(certId) ‘Fetch the certificate chain to be passed to the server

‘ Most applications do not capture signature image and hence the following client side steps are not needed for them.
clientSigObj.PhysicalSignatureUsage = 2 ‘Optional. Choosing live signature image capture
signatureImageData = clientSigObj.ReadAllBase64(clientSigObj.CaptureLiveSignature) ‘Capture signature and read the base64
encoded signature image and pass it to the server
signatureImageId = clientSigObj.SignatureImageId ‘Pass the signature image id to the server
```

Switch to server side:

```
Set serverSigObj = CreateObject("XMLSign.Signature") ‘Create SecureXML object

‘ These three step are required only if a signature image was captured on the client side
serverSigObj.signatureImageId = signatureImageId ‘Get it from client side
serverSigObj.PhysicalSignatureB64Str = signatureImageData ‘Get it from client side
serverSigObj.PhysicalSignatureUsage = 3

‘ Set the certificate chain if a certificate selection was made on the client side prior to coming here
serverSigObj.SignerCertificateChain = certChain ‘Get it from client side

‘ CRL checking and CRL inclusion before digest calculation make sense only if the certificate chain was set above
serverSigObj.CrlChecking = 1
serverSigObj.IncludeCRLInSignature = 1

‘The following code assumes C:\temp\signature.tmpl contains the XML template that needs to be signed and it has a <signature>
‘element with Id = “MySignature”. The XML template must contain <SignaureMethod Algorithm=”…” /> element if
‘ SignerCertificateChain was not set above otherwise GetSignedInfoDigest will report an error.
‘ The first parameter to GetSignedInfoDigest method may be null if either AttachedObjects or DetachedObjects properties have
‘ been set prior to coming here. In which case, SecureXML will prepare the XML envelope and include the signed references
‘ before digest calculation
signedInfoDigest = serverSigObj.GetSignedInfoDigest(serverSigObj.ReadAll("C:\temp\signature.tmpl"), "MySignature")
```

Back to client side:

```
certData = clientSigObj.GetX509Certificate(certId) ‘Fetch the base64 encoded certificate data
‘ The certData parameter below could be null. If it is null, SecureXML will display the certificate selection dialog box.
b64SigValXml = clientSigObj.SignSignedInfoDigest(certData, signedInfoDigest)
```

Back to server side:

```
signedXML = serverSigObj.ApplySignatureValue(b64SigValXml) ‘Get it from client side
```

Now the signedXML has the signed xml.

Working with Certificates

The SecureXML Digital Signature provides an easy to use interface for working with cryptographic certificates under Windows. Before invoking any methods on this object, it is a good idea to first check the CertificateCount value. A value of zero indicates that the current user has no certificates installed in the machine where the object resides.

Certificate Dialog Box Behavior

If an active certificate is not selected programmatically before invoking any of the signature creation methods, a certificate selection dialog box is presented to the user. The certificate selection dialog box is supported only on Windows platform.

Many times you would want to select active certificate programmatically without having the user manually select it. This would certainly be the case if the user has only one certificate installed on the computer. Also if a smart card or other hardware token is being used for private key storage, the user will have to go through two dialog boxes: one for certificate selection and one for providing the password to the hardware device for getting access to the private key. The following section provides an example of how one can build a table in an HTML page where users can select their certificate by clicking in a radio button. One can of course not display any such interface and simply rely of user preference etc. for selecting certificates for silent operation.

Relevant properties and methods

CertificateCount, GetCertificateInfo, SetActiveCertificate, SelectActiveCertificate

You can get a list of available certificates by doing the following:

```
For index = 0 to (CertificateCount – 1)
        For certDataType = 1 to 5
                CertData[index, certDataType] = GetCertificateInfo(index, certDataType)
        End For
End For
```

The following table provides the various certificate data returned for each of the certDataType values:

certDataType	Return Value
1	Certificate Serial Number
2	Certificate Issuer Name
3	Certificate Subject Name
4	Certificate Expiration Date
5	Certificate Subject Short Name

SetActiveCertificate takes a certificate serial number as an input and makes the certificate with the matching serial number the active certificate for all subsequent calls to Sign/SignXMLStr/SignFile/SignXMLEnveloped/SignHTML methods.

SelectActiveCertificate pops a window with the list of valid certificates and the user can select the certificate for all subsequent calls to Sign/SignXMLStr/SignFile/SignXMLEnveloped/SignHTML methods.

Here is the VB Script code taken directly from the online demo application showing how to use certificates for signing a file:

```
Dim certSerialNumber, varHtml, CurVersion
‘ This function gets invoked when the user clicks on one of the certificate selection radio buttons which are generated during a
‘call to loadForm() function

Function setCert(snum)
        certSerialNumber = snum
End Function
```

```
‘ loadForm() is called on page load
Function loadForm()
Dim Element, CertIndex, retval, count, SerialNo, IssuedBy, IssuedTo, ExpiryDate
varHtml = "<TABLE BORDER=1 CELLPADDING=0><TR><TD> </TD><TD>Issued To</TD><TD>Issued
By</TD><TD>Exp.Date</TD></TR>"
count = SigObj.CertificateCount

If count <> 0 Then              ‘ There is at least one certificate present
        For CertIndex = 0 to (count-1)
                SerialNo = SigObj.GetCertificateInfo(CertIndex, 1)
                IssuedBy = SigObj.GetCertificateInfo(CertIndex, 2)
                IssuedTo = SigObj.GetCertificateInfo(CertIndex, 3)
                ExpiryDate = SigObj.GetCertificateInfo(CertIndex, 4)
                If count = 1 Then
                        varHtml = varHtml & "<TR><TD><INPUT TYPE=radio onclick=setCert(&quot;" & SerialNo &
                        "&quot;) NAME=&quot;certId&quot; VALUE=&quot;" & SerialNo & "&quot; CHECKED>" &
                        "</TD><TD>" & IssuedTo & "</TD><TD>" & IssuedBy & "</TD><TD>" & ExpiryDate &
                        "</TD></TR>"
                        setCert(SerialNo)
                else
                        varHtml = varHtml & "<TR><TD><INPUT TYPE=radio onclick=setCert(&quot;" & SerialNo &
                        "&quot;) NAME=&quot;certId&quot; VALUE=&quot;" & SerialNo & "&quot;>" &
                        "</TD><TD>" & IssuedTo & "</TD><TD>" & IssuedBy & "</TD><TD>" & ExpiryDate &
                        "</TD></TR>"
                End If
        Next
        varHtml = varHtml & "</TABLE>"
else
        varHtml ="<br><font color=red size=3>You must have a PKI certificate with access to private key to sign
        documents.<br><a href=https://www.infomosaic.net/certificate>Please click here to get one for free</a></font>"
End if

 ‘ The following will dynamically create the radio button elements for the certificates currently available
CertData.innerHTML = varHtml
End Function

Function ProcessSignature()
        Dim retval, storedString, sigID, xmlfile, xmlfs, xmlSignature, var1, reason
        retval = 1
        var1 = ""
        if (certSerialNumber = "") Then
                MsgBox("Please select a signature key")
                Exit Function
        end if
        retval = SigObj.SetActiveCertificate(certSerialNumber)
        if (UserInfo.EnvelopingFlag.Checked = True) Then
          SigObj.EnvelopingFlag = 1
        else
          SigObj.EnvelopingFlag = 0
        end if
        sigID = “MySignature”
        SigObj.SignatureID(0) = sigID
        storedString = SigObj.Sign(UserInfo.file_name.value)
        if (storedString <> "") Then 'Signature successful
                retval = SigObj.SaveXMLSignature(UserInfo.out_file_name.value)
        end if
        if (SigObj.GetLastError <> 0) Then 'Signing failed
                var1 = var1 & "<br>XML Signature Failed, Reason = " & SigObj.GetError
                UserInfo.SignatureStatus.value = False
                UserInfo.SignatureResult.value = var1
                Exit Function
        end if
```

```
        'Signature successful: save the signature in the destination file
        var1 = var1 & "<br>XML Signature Successful"
        var1 = var1 & "<br>Signature ID = " & sigID
        UserInfo.SignatureStatus.value = True
        UserInfo.SignatureResult.value = var1
End Function

<form name="UserInfo" Action="SignatureResult.asp" method=post>
 <input name=inputFileName type=hidden value="">
 <input name=outputFileName type=hidden value="">
 <input name=SignatureResult type=hidden value="">
 <input name=SignatureStatus type=hidden value="">
 <table> <tbody>
    <tr> <td ><font size=3  color=white><b>XML Signature: Sign document</b></font> </td> </tr>
    <tr>
      <td colspan=3><font color=white>Your are going to sign <% Response.Write(Request("inputFileName"))%></font> </td>
     </tr><tr>
       <td colspan=3><font color=white>The signature will be saved as <% Response.Write(Request("outputFileName"))%>
</font> </td>
    </tr> <tr>
     <td colspan=2> <font color=white size=3> Include document in the signature file</font> </td>
     <td> <input type="checkbox" CHECKED name="EnvelopingFlag" onclick="EnvelopingFlag_Clicked()"></td>
    </tr><tr>
      <td colspan=3><label id="selectCertMsg" style="DISPLAY: none"><font color=white size=3>Please select a key for your
signature:</font></label></td>
    </tr> <tr>
       <td colspan=3><label id="usingCertMsg" style="DISPLAY: none"><font color=white size=3>Using the following key for
your signature:</font></label></td>
     </tr><tr> <td colspan=3> <span id="CertData"> Certificates  </span> </td> </tr>
     <tr>
       <td> <input onClick="ViewCert()" type=button value="View Certificate" id=button2 name=button2> </td>
       <td colspan=2> <input onClick="return ProcessSignature()" type=submit value="Sign Document" name="button"></td>
     </tr></tbody>
  </table>
</form>
```

Certificate Validation

There are several different ways to verify the validity of a certificate. They are based on Online CRL, Offline CRL, CAM Server and OCSP Responder. By default SecureXML does minimal certificate validation. The root issuer of the certificate being used must be in the Windows root certificate store. For all other validations the corresponding validation properties must be set.

The following matrix shows how you can get the certificate validation status at signature creation time when you are verifying a signature created earlier:

Validation Method	*Response Included?*	*Has Certificate Expired?*	*Validity at signing time*
CRL	Yes	Yes	Invoke SigCertStatus with CrlChecking = 0 time = ""
CRL	Yes	No	Invoke SigCertStatus with CrlChecking = 0 or 1 time = ""
CRL	No	Yes	Invoke SigCertStatus with CrlChecking = 1 time = "" if response is valid then status unknown
CRL	No	No	Invoke SigCertStatus with CrlChecking = 1 time = ""
CAM	Yes	Yes	Valid
CAM	Yes	No	Valid
CAM	No	Yes	Unknown
CAM	No	No	Unknown
None	No	Yes	Invoke SigCertStatus with CrlChecking = 1 time = "". If response is valid, status is expired/unknown because CA is not required to report revocation status of an expired certificate else revoked.
None	No	No	Invoke SigCertStatus with CrlChecking = 1 time = ""

The following matrix shows how you can get the current certificate validation status:

Validation Method	*Response Included?*	*Has Certificate Expired?*	*Validity at current time*
CRL or None	N/A	Yes	Invoke SigCertStatus with CrlChecking = 1, time = localtime. If response is valid, status is expired/unknown because CA is not required to report revocation status of an expired certificate else revoked
CRL or None	N/A	No	Invoke SigCertStatus with CrlChecking = 1, time = localtime
CAM	N/A	Yes	Expired
CAM	N/A	No	Invoke SigCertStatus with CrlChecking = 1, time = localtime

OCSP Certificate Validation

The following object properties are relevant to using OCSP based certificate validation:

- UseOcsp
- OcspTrustedRespSignerCertPath
- OcspReqSignerPFXCertPath
- OcspReqSignerPFXCertPassword
- OcspResponderURL
- OcspTextResponse
- OcspB64Response
- CertRevocationDate
- IncludeOcspResponse

Before you can use OCSP certificate validation method, please make sure that ocspAX.dll is present and registered. The following VBScript illustrates the use of OCSP with SecureXML:

```
Set SigObj = CreateObject("XMLSign.Signature")

// If your OCSP responder requires signed OCSP requests (as is the case for Identrus compliance), provide the path to the
// PFX/P12 file containing the signer certificate and private key and set the password:

SigObj.OcspReqSignerPFXCertPath = "C:\ocspAX\DSTTest\signer.pfx"
SigObj.OcspReqSignerPFXCertPassword = "dstacestest"

// If you want to restrict the trusted OCSP response signer to a given CA certificate, please provide the path to the PEM file
// containing the entire certificate chain for the OCSP response signer certificate.

SigObj.OcspTrustedRespSignerCertPath = "C:\ocspAX\DSTTest\responderca.pem"

// If the responder URL contained in the certificate's AIA extension is not to be used, provide the URL for the OCSP responder
SigObj.OcspResponderURL = http://ocsp.digsigtrust.com

// Enable OCSP based certificate validation
SigObj.UseOcsp = 1

// The CertValidationTransactionId can't be set before setting UseOcsp = 1
SigObj.CertValidationTransactionId = "1234"

// Include both text and binary responses in the current signature
SigObj.IncludeOcspResponse = 3

// Sign a simple string
signedXML = SigObj.SignDataStr("This is a test")
SigObj.SaveXMLStr signedXML, "C:\temp\s1Ocsp.xml"
SigObj.CertValidationTransactionId = "12345"
SigObj.VerifyXMLStr signedXML
result = SigObj.SigCertStatus(0,"",1)
MsgBox("Verification Result = " & result & " " & SigObj.GetError)
MsgBox(SigObj.OcspTextResponse)
MsgBox(SigObj.CertValidationTransactionId)
```

Working with PFX/P12 or PEM Files and Data

Following method is provided for creating a signature using the private key stored in a PFX/P12 or PEM (support only on Windows) file:

SetActivePFXFileCert
SetActivePEMFileCert

The following methods are provided for exporting certificates along with private key to a PFX file:

PFXExportCertificate
PFXExportActiveCertificate

Creating signature using certificate stored in a PFX/P12/PEM file:

SetActivePFXFileCert and SetActivePEMFileCert methods take the PFX/PEM file name and the password used during creating the PFX/PEM file as two input parameters and returns the corresponding Base64 encoded X509 certificate, which can be used to extract signer information. It sets this PFX/PEM key to be the active certificate for all subsequent calls to all signature creation functions. A call to either SetActiveCertificate or SelectActiveCertificate must be made in order to change the active signer certificate. You may call SetActivePFXFileCert / SetActivePEMFileCert again to change the active certificate to another PFX/PEM file based certificate.

Exporting certificates in PFX format from the local certificate store:

PFXExportCertificate takes the certificate serial number (certId) and a password as input parameter and returns a file path for a temp file where the exported certificate has been stored.

PFXExportActiveCertificate saves you the trouble of providing a certificate serial number. Once the user has selected a certificate for signing, you can export that particular certificate by supplying a password parameter. Internally this function takes the certificate serial number of the currently active certificate and calls PFXExportCertificate. The return value is again file path for temp file where the exported certificate has been stored.

Creating Non-PKI/HMAC or Password based XML Signature

Two properties need to be set before invoking any of the signature creation methods in order to create password based XML Signature: UseHMAC and HMACPassword. Set UseHMAC equal to 1 and assign HMACPassword to your password. All signature creation methods (except the **Sign** method when invoked with **EnvelopingFlag** set to either 0 or 1) check for the UseHMAC property and if it is set and if HMACPassword is not NULL, produce HMAC XML Signature. All signature verification functions check for HMACPassword before verifying any signature and if any signature has HMAC as the signature method, they use HMACPassword as the password or the HMAC key for verification.

Working with Physical Electronic Signature

This feature is supported only on Windows.

With the release of version 2.0.125.26, support for physical signature capture and addition to enveloped XML signatures have been added. It provides additional functionality for document signing where a physical signature may need to be produced in printed form. It also provides a better feel for online document signing and verification.

With SecureXML Digital Signature, using physical signature is very easy. In addition to any other setup for your signature, all you need to do is set the PhysicalSignatureUsage property to either 1 (to add file signature) or 2 (to capture live signature).

The following will present a signature capture window when Sign() is invoked with EnvelopingFlag set to 2 (for enveloped signature) and since all other signature creation methods (SignXMLStr, SignHTML, SignXMLEnveloped, SignFile) always create enveloped signature, there is nothing additional one needs to do.

SigObj.PhysicalSignatureUsage = 2

The following requires you to provide full path for the signature image file:

SigObj.PhysicalSignatureUsage = 1

Now provide the path to your signature image file:

SigObj.PhysicalSignatureFile = “C:\mySignature.bmp”

Image formats supported include bmp, jpg, gif and png.

Making Your Application DoD PKI Compliant

With the release of version 2.1.129.31, SecureXML provides an easy way to comply with US Department of Defence (DoD) PKI standard administered by JITC. All the application needs to do after creating an instance of the SecureXML Signature object is to set the DoDCompliance property to 1. It enables all the relevant certificate checking. In addition to the setting DoDCompliance, the application may also need to provide a list of trusted root certificates and set CertificateTrustExplicit property to 1.

The following are the relevant object properties for DoD Compliance:

- DoDCompliance
- TrustedRoots
- CRLLocation
- CertificatePolicy
- AuthorityConstrainedPolicy
- UserConstrainedPolicy
- CertificatePolicyExplicit

Here is what all this will look like in an application:

```
Dim rootList(2) ‘ if there are two trusted root certificates
SigObj = CreateObject(“XMLSign.Signature”)
SigObj.DoDCompliance = 1
rootList(0) = “http://www.army.mil/TrustedRoot.cer”
rootList(1) = “C:\trust\root.cer”
SigObj.TrustedRoots = rootList

‘ From now on all calls to sign and verify functions will check for certificate validity as per DoD PKI standards.
```

Using Encryption

With the release of version 2.1.129.31, SecureXML provides an easy way to encrypt data either stored in a buffer or a file. A typical use case would be when a signed document must be kept confidential and is for only a select few to see. The following methods and properties are relevant to this feature:

Object methods:

- EncryptStr
- DecryptStr
- EncryptFile
- DecryptFile

Object properties:

- RecipientCertificates
- RecipientCertificateFiles

Please note a license file with <Encrypt> and <Decrypt> element contents set to 1 is required in order to use the encryption methods. This is due to US government export control on encryption technology. Currently the encryption supports 128 bit RSA-RC4 encryption only. In addition to the 128-bit key a 12 byte salt is also used. Future version of SecureXML will provide functionality to set additional encryption algorithms and change salt size etc.

The following code shows how you can use the above methods and properties to encrypt and decrypt data:

```
Set SigObj = CreateObject("XMLSign.Signature")
res = SigObj.EncryptStr("This is my input. This is good input")
MsgBox(res)
SigObj.SaveXMLStr res, "C:\temp\encrpted.txt"
res = SigObj.DecryptStr(res)
MsgBox(res)

Set SigObj = Nothing

Dim certList(2)
Set SigObj = CreateObject("XMLSign.Signature")
certList(0) = "C:\temp\recp1.crt"          'DER Encoded binary X509 certificate file
certList(1) = "C:\temp\recp2.cer"          'Base64 encoded (without any extra lines such as BEGIN CERTIFICATE) X509 file
SigObj.RecipientCertificateFiles = certList

'Alternately you could have the Base64 encoded X509 certificate in a buffer and use the following
' cert1 = SigObj.ReadAll("C:\temp\recp1.cer")
' cert2 = SigObj.ReadAll("C:\temp\recp2.cer")
' certList(0) = cert1
' certList(1) = cert2
'SigObj.RecipientCertificates = certList

res = SigObj.EncryptStr("This is my input. This is good input")
if (res = "") then
        MsgBox(SigObj.GetLastError)
else
        MsgBox(res)
end if
res = SigObj.DecryptStr(res)            'You will get blank if at least one of the certificates in certList is not your own
                                        ' You can't decrypt something without access to the private key
if (res = "") then
        MsgBox(SigObj.GetLastError)
else
        MsgBox(res)
end if
```

```
Set SigObj = Nothing

Set SigObj = CreateObject("XMLSign.Signature")
SigObj.RecipientCertificateFiles = certList
res = SigObj.EncryptFile("c:\temp\myfile.txt", "C:\temp\encmyfile.txt")
if (res = "") then
        MsgBox(SigObj.GetLastError)
else
        MsgBox(res)
end if
res = SigObj.DecryptFile("C:\temp\ encmyfile.txt", "C:\temp\decryptedmyfile.txt")
if (res = "") then
        MsgBox(SigObj.GetLastError)
else
        MsgBox(res)
end if
```

Using Netscape Certificate Store

In order to use Netscape certificate store for signer certificate selection, two things need to be done:

1. Set the Netscape Certificate Store password (if the store is password protected) by setting the NetscapeStorePassword object property to the right value, and
2. Invoke SetStoreName method with “Netscape” (case sensitive) as the input parameter.

Every thing else is transparent for the end application.

SecureXML Application Programming Interface (API) Reference

Object Properties

AddWindowImage

Read / Write	Read / Write
IDL File Declaration	[propget, id(68), helpstring("property AddWindowImage")] HRESULT AddWindowImage([out, retval] BOOL *pVal); [propput, id(68), helpstring("property AddWindowImage")] HRESULT AddWindowImage([in] BOOL newVal);

Java Interface	public int getAddWindowImage();
	public void setAddWindowImage(int lastParam);
	int curAddWinImgVal = sigObj.getAddWindowImage();
	sigObj.setAddWindowImage(1); // Enable window image capture feature

C Interface	ISignature_get_AddWindowImage(ISignature *pSig, BOOL *pVal); ISignature_put_AddWindowImage(ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_AddWindowImage(BOOL *pVal); pSig->put_AddWindowImage(BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.AddWindowImage = 1 pVal = sigObj.AddWindowImage
Supported Platforms	Windows Only

This property enables the window image capture feature of SecureXML. The default value is zero or disabled. Please note that the image captured is limited to the visible area of the foreground window. If you use this feature from a web page, which has additional data accessible by scrolling up or down, the captured image will not include currently invisible data. Depending on the screen resolution and window size, the size of the resulting XML Signature could be large. The image is included as base64 encoded data of the corresponding image in PNG format.

Parameters:

pVal	Returns the current value of the AddWindowImage property.
newVal	Sets the value of the AddWindowImage property to newVal. A value of 0 (zero) disables this feature. Any non-zero value enables it.

GetLastError can return the following error codes:

None.

AgencyId

Read / Write	Both
IDL File Declaration	[propget, id(109), helpstring("property AgencyId")] HRESULT AgencyId([out, retval] BSTR* pVal); [propput, id(109), helpstring("property AgencyId")] HRESULT AgencyId([in] BSTR newVal);
Java Interface	public String getAgencyId (); public void setAgencyId (String lastParam); String curAgencyIdVal = sigObj.getAgencyId (); sigObj.setAgencyId ("AA Agency Id"); // Set AgencyId to 'AA Agency Id'
C Interface	ISignature_get_AgencyId (ISignature *pSig, BSTR *pVal); ISignature_put_AgencyId (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_AgencyId (BSTR *pVal); pSig->put_AgencyId (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.AgencyId= "AA Agency Id" pVal = sigObj.AgencyId
Supported Platforms	All

If UseCam property is set, the value of AgencyId is used for submitting the certificate validation requests to the CAM server.

Parameters:

newVal	Sets the AgencyId to newVal.
pVal	Returns the current settings for AgencyId.

AllowedCertIssuerNames

Read / Write	Write Only
IDL File Declaration	[propput, id(120), helpstring("property AllowedCertIssuerNames")] HRESULT AllowedCertIssuerNames([in] VARIANT newVal);
Java Interface	public void setAllowedCertIssuerNames(String [] lastParam); String [] allowedCertIssuerNameList = new String[3]; allowedCertIssuerNameList [0] = "Infomosaic Corporation"; allowedCertIssuerNameList [1] = "DoD PKI Root Issuer"; allowedCertIssuerNameList [2] = "ACES TEST CERT ISSUER"; sigObj.setAllowedCertIssuerNames (allowedCertIssuerNameList);
C Interface	ISignature_put_AllowedCertIssuerNames (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_AllowedCertIssuerNames (VARIANT newVal);

VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim allowedCertIssuerNameList (3)
	allowedCertIssuerNameList [0] = "Infomosaic Corporation"
	allowedCertIssuerNameList [1] = "DoD PKI Root Issuer"
	allowedCertIssuerNameList [2] = "ACES TEST CERT ISSUER"
	sigObj.AllowedCertIssuerNames = allowedCertIssuerNameList
	Or
	string []allowedCertIssuerNameList = new String[3];
	allowedCertIssuerNameList [0] = "Infomosaic Corporation";
	allowedCertIssuerNameList [1] = "DoD PKI Root Issuer";
	allowedCertIssuerNameList [2] = "ACES TEST CERT ISSUER";
	sigObj.AllowedCertIssuerNames = allowedCertIssuerNameList;

Supported Platforms	All

When this property is set, SecureXML displays only the certificates issued by the given issuers. CertificateCount returns the number of certificates that match this restriction. On non-Windows platforms, even though there is no certificate selection window GUI, the programmatic interface to getting certificates reflects the changes as per this property setting.

Parameters:

newVal The list of certificate issuer to use for restricting the list of available certificates for use.

AttachedObjects

Read / Write	Write Only
IDL File Declaration	[propput, id(100), helpstring("property AttachedObjects")] HRESULT AttachedObjects([in] VARIANT newVal);

Java Interface	public void setAttachedObjects(String [] lastParam);
	String [] attachedObjectList = new String[3];
	attachedObjectList[0] = "http://www.infomosaic.com/index.html";
	attachedObjectList[1] = "file:///c:/temp/mydata.doc";
	attachedObjectList[2] = "G:\temp\mySpreadSheet.xls";
	sigObj.setAttachedObjects(attachedObjectList);

C Interface	ISignature_put_AttachedObjects(ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_AttachedObjects (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim attachedObjectList(3) attachedObjectList[0] = "http://www.infomosaic.com/index.html"; attachedObjectList[1] = "file:///c:/temp/mydata.doc"; attachedObjectList[2] = "G:\temp\mySpreadSheet.xls"; sigObj.AttachedObjects = attachedObjectList Or string [] attachedObjectList = new String[3]; attachedObjectList[0] = "http://www.infomosaic.com/index.html"; attachedObjectList[1] = "file:///c:/temp/mydata.doc"; attachedObjectList[2] = "G:\temp\mySpreadSheet.xls"; sigObj.AttachedObjects = attachedObjectList
Supported Platforms	All

If AttachedObjects is non-null, all the objects pointed to by AttachedObjects are included inside the current signature (as in an enveloping signature) as object references to base64 encoded data. If any of these objects are not accessible, the signature creation will fail.

Parameters:

newVal The given objects to be included in an enveloping manner for the current signature.

AuthorityConstrainedPolicy

Read / Write	Read Only
IDL File Declaration	[propget, id(72), helpstring("property AuthorityConstrainedPolicy")] HRESULT AuthorityConstrainedPolicy([out, retval] VARIANT *pVal);
Java Interface	public String [] getAuthorityConstrainedPolicy(); String [] authConstPolVar = sigObj.getAuthorityConstrainedPolicy();
C Interface	ISignature_get_AuthorityConstrainedPolicy(ISignature *pSig, VARIANT *authConstPolSet);
C++ Interface	pSig->get_AuthorityConstrainedPolicy (VARIANT *authConstPolSet);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Set authConstPolSet = sigObj.AuthorityConstrainedPolicy Or string [] authConstPolSet = (string [])sigObj.AuthorityConstrainedPolicy;
Supported Platforms	All

This property returns the authority constrained policy set after a certificate chain policy verification.

Parameters:

pVal Returns the authority constrained policy set after a certificate chain policy verification as a VARIANT.

Base64DecodeXML

Read / Write	Read / Write
IDL File Declaration	[propget, id(7), helpstring("property Base64DecodeXML")] HRESULT Base64DecodeXML([out, retval] BOOL *pVal); [propput, id(7), helpstring("property Base64DecodeXML")] HRESULT Base64DecodeXML([in] BOOL newVal);

Java Interface	public int getBase64DecodeXML();
	public void setBase64DecodeXML (int lastParam);
	int curBase64DecodeXMLVal = sigObj.getBase64DecodeXML ();
	sigObj.setBase64DecodeXML (1); // Enable base64 Decoding feature

C Interface	ISignature_get_Base64DecodeXML (ISignature *pSig, BOOL *pVal); ISignature_put_Base64DecodeXML (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_Base64DecodeXML (BOOL *pVal); pSig->put_Base64DecodeXML (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.Base64DecodeXML = 1 pVal = sigObj.Base64DecodeXML
Supported Platforms	All

If XML contains native characters, sometimes when you save the signed XML to a file, it can undergo character encoding transformation, which leads to a failure during signature verification. The Base64DecodeXML property is provided to avoid such problems. When Base64DecodeXML is enabled, all signature creation methods base64 decode the input string before attempting to sign it. In other words, if Base64Decode is enabled, all input for enveloped signature must be base64 encoded by the caller. Signature verification methods do not get affected by this property and continue to expect the input signed XML to be unencoded. Please refer to Base64EncodeXML property for how to provide base64 encoded signed XML to verification methods.

Parameters:

pVal	Returns the current value of the Base64DecodeXML property.
newVal	Sets the value of the Base64DecodeXML property to newVal. A value of 0 (zero) disables this feature. Any non-zero value enables it.

Base64EncodeXML

Read / Write	Read / Write
IDL File Declaration	[propget, id(6), helpstring("property Base64EncodeXML")] HRESULT Base64EncodeXML([out, retval] BOOL *pVal); [propput, id(6), helpstring("property Base64EncodeXML")] HRESULT Base64EncodeXML([in] BOOL newVal);

Java Interface	public int getBase64EncodeXML();
	public void setBase64EncodeXML (int lastParam);
	int curBase64EncodeXMLVal = sigObj.getBase64EncodeXML ();
	sigObj.setBase64EncodeXML (1); // Enable base64 encoding feature

C Interface	ISignature_get_Base64EncodeXML (ISignature *pSig, BOOL *pVal); ISignature_put_Base64EncodeXML (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_Base64EncodeXML (BOOL *pVal); pSig->put_Base64EncodeXML (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.Base64EncodeXML = 1 pVal = sigObj.Base64EncodeXML
Supported Platforms	All

If XML contains native characters, sometimes when you save the signed XML to a file, it can undergo character encoding transformation, which leads to a failure during signature verification. The Base64EncodeXML property is provided to avoid such problems. When Base64EncodeXML is enabled, all signature creation methods base64 encode the signed XML before returning it to the caller. Signature verification methods treat the input XML to be base64 encoded and they first do a base64 decode before attempting to verify the signature.

Parameters:

pVal	Returns the current value of the Base64EncodeXML property.
newVal	Sets the value of the Base64EncodeXML property to newVal. A value of 0 (zero) disables this feature. Any non-zero value enables it.

CamServerHost

Read / Write	Both
IDL File Declaration	[propget, id(106), helpstring("property CamServerHost")] HRESULT CamServerHost([out, retval] BSTR* pVal); [propput, id(106), helpstring("property CamServerHost")] HRESULT CamServerHost([in] BSTR newVal);
Java Interface	public String getCamServerHost (); public void setCamServerHost (String lastParam); String curCamServerHostVal = sigObj.getCamServerHost (); sigObj.setCamServerHost ("testserver.infomosaic.net"); // Set CamServerHost to 'testserver.infomosaic.net'
C Interface	ISignature_get_CamServerHost (ISignature *pSig, BSTR *pVal); ISignature_put_CamServerHost (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_CamServerHost (BSTR *pVal); pSig->put_CamServerHost (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CamServerHost= "testserver.infomosaic.net" pVal = sigObj.CamServerHost
Supported Platforms	All

If UseCam property is set, the CamServerHost is contacted for all certificate validation requests.

Parameters:

newVal	Sets the CamServerHost to newVal.
pVal	Returns the current settings for CamServerHost.

CamServerPort

Read / Write	Both
IDL File Declaration	[propget, id(107), helpstring("property CamServerPort")] HRESULT CamServerPort([out, retval] USHORT* pVal); [propput, id(107), helpstring("property CamServerPort")] HRESULT CamServerPort([in] USHORT newVal);

Java Interface	public String getCamServerPort ();
	public void setCamServerPort (short lastParam);
	short curCamServerPortVal = sigObj.getCamServerPort ();
	sigObj.setCamServerPort (8181); // Set CamServerPort to 8181

C Interface	ISignature_get_CamServerPort (ISignature *pSig, USHORT *pVal); ISignature_put_CamServerPort (ISignature *pSig, USHORT newVal);
C++ Interface	pSig->get_CamServerPort (USHORT *pVal); pSig->put_CamServerPort (USHORT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CamServerPort= 8181 pVal = sigObj.CamServerPort
Supported Platforms	All

If UseCam property is set, the CamServerHost on port CamServerPort is contacted for all certificate validation requests. The default port number is 7777 if not explicitly set by the application.

Parameters:

newVal	Sets the CamServerPort to newVal.
pVal	Returns the current settings for CamServerPort.

CamValidationResponse

Read / Write	Read Only
IDL File Declaration	[propget, id(110), helpstring("property CamValidationResponse")] HRESULT CamValidationResponse([out, retval] BSTR* pVal);
Java Interface	public String getCamValidationResponse (); String curCamValidationResponseVal = sigObj.getCamValidationResponse ();
C Interface	ISignature_get_CamValidationResponse (ISignature *pSig, BSTR *pVal);
C++ Interface	pSig->get_CamValidationResponse (BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CamValidationResponse
Supported Platforms	All

It returns the response received from the CAM server during the last certificate validation request.

Parameters:

pVal	Response received from the CAM server during the last certificate validation request

CanonicalizationMethod

Read / Write	Write Only
IDL File Declaration	[propput, id(124), helpstring("property CanonicalizationMethod")] HRESULT CanonicalizationMethod([in] ULONG newVal);
Java Interface	public void setCanonicalizationMethod(int lastParam); sigObj.setCanonicalizationMethod (0); // Set CamServerPort to 8181
C Interface	ISignature_put_CanonicalizationMethod (ISignature *pSig, ULONG newVal);
C++ Interface	pSig->put_CanonicalizationMethod (ULONG newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CanonicalizationMethod = 0
Supported Platforms	All

For signature creation methods where SecureXML prepares the <Signature> element from scratch, you can control whether comments are included in the canonicalization or not. By default, SecureXML uses http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments as the canonicalization method. Set CanonicalizationMethod to 0 if you want SecureXML to use http://www.w3.org/TR/2001/REC-xml-c14n-20010315 instead. Values 2 and 3 set the Canonicalization Algorithm to Exclusive Canonicalization methods http://www.w3.org/2001/10/xml-exc-c14n# and http://www.w3.org/2001/10/xml-exc-c14n#WithComments respectively.

Parameters:

newVal	Sets the CanonicalizationMethod to newVal. If newVal = 0, canonicalization method is set to http://www.w3.org/TR/2001/REC-xml-c14n-20010315 and if newVal = 1, it is set to http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments (default). Values 2 and 3 set the Canonicalization Algorithm to http://www.w3.org/2001/10/xml-exc-c14n# and http://www.w3.org/2001/10/xml-exc-c14n#WithComments respectively

CapturedSignatureFile

Read / Write	Read Only
IDL File Declaration	[propget, id(14), helpstring("property CapturedSignatureFile")] HRESULT CapturedSignatureFile([out, retval] BSTR *pVal);
Java Interface	public String getCapturedSignatureFile(); String signatureImageFile = sigObj.getCapturedSignatureFile();
C Interface	ISignature_get_CapturedSignatureFile(ISignature *pSig, BSTR *pVal);
C++ Interface	pSig->get_CapturedSignatureFile(BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CapturedSignatureFile
Supported Platforms	Windows Only

Returns the full path of the signature image file stored on the disk after a successful signature image capture. This file is automatically deleted when a new image is captured or when the SecureXML Digital Signature object is destroyed.

Parameters:

pVal	Returns the full path of the signature image file stored on the disk after a successful signature image capture.

CertExpiry

Read / Write	Read Only
IDL File Declaration	[propget, id(32), helpstring("property CertExpiry")] HRESULT CertExpiry([in] BSTR sigId, [out, retval] BSTR *pVal);
Java Interface	public String getCertExpiry(String sigId); String certificateExpiration = sigObj.getCertExpiry(sigId);
C Interface	ISignature_get_CertExpiry(ISignature *pSig, BSTR sigId, BSTR *pVal);
C++ Interface	pSig->get_CertExpiry(BSTR sigId, BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CertExpiry(sigId)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default). It returns the certificate expiration date of the X509Certificate used to create the signature with signature id = sigId.

Parameters:

sigId	This is the signature ID of a signature which was verified prior to invoking CertExpiry(). A signature ID is obtained by calling SignatureID() property getter.
pVal	On return *pVal is set to the certificate expiration date for the signer's certificate in the given signature. If a signature with signature id = sigId is not found *pVal is set to "".

GetLastError can return the following error codes:

SIG_NOT_FOUND	A signature with matching sigId was not found in the signature just verified.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

CertificateChainValidation

Read / Write	Write only.
IDL File Declaration	[propput, id(75), helpstring("property CertificateChainValidation")] HRESULT CertificateChainValidation([in] BOOL newVal);
Java Interface	public void setCertificateChainValidation (int lastParam); sigObj.setCertificateChainValidation (1); // Enable Certificate Chain Validation
C Interface	ISignature_put_CertificateChainValidation (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_CertificateChainValidation (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CertificateChainValidation = 1
Supported Platforms	All

This property enables or disables certificate chain validation. The default value is 0 (zero) or disabled. If it is set to a non-zero value, certificate chain validation gets enabled for all calls to SecureXMLVerify() and all signature creation methods.

If DoDCompliance is set to one, any attempt to set this property to zero is ignored.

Parameters:

newVal Sets the CertificateChainValidation property to newVal

---

CertificateCount

Read / Write	Read Only
IDL File Declaration	[propget, id(26), helpstring("property CertificateCount")] HRESULT CertificateCount([out, retval] long *pVal);
Java Interface	public int getCertificateCount(); int certCount = sigObj.getCertificateCount();
C Interface	ISignature_get_CertificateCount(ISignature *pSig, long *pVal);
C++ Interface	pSig->get_CertificateCount(long *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CertificateCount
Supported Platforms	All

Returns the total number of valid certificates in the current certificate store.

Parameters:

pVal On return *pVal is set to the number of valid certificates in the current certificate store.

CertificatePathLengthChecking

Read / Write	Write only.
IDL File Declaration	[propput, id(76), helpstring("property CertificatePathLengthChecking")] HRESULT CertificatePathLengthChecking([in] BOOL newVal);
Java Interface	public void setCertificatePathLengthChecking (int lastParam); sigObj.setCertificatePathLengthChecking (1); // Enable Certificate Path Length Checking
C Interface	ISignature_put_CertificatePathLengthChecking (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_CertificatePathLengthChecking (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CertificatePathLengthChecking = 1
Supported Platforms	All

This property enables or disables certificate path length validation during certificate chain validation. The default value is 0 (zero) or disabled. If it is set to a non-zero value, certificate path length validation gets enabled for all calls to SecureXMLVerify() and all signature creation methods.

If DoDCompliance is set to one, any attempt to set this property to zero is ignored.

Parameters:

newVal Sets the CertificatePathLengthChecking property to newVal

---

CertificatePolicy

Read / Write	Write Only
IDL File Declaration	[propput, id(71), helpstring("property CertificatePolicy")] HRESULT CertificatePolicy([in] VARIANT newVal);

Java Interface	public void setCertificatePolicy(String [] lastParam);
	String [] policyList = new String[3];
	policyList[0] = "2.16.840.1.101.3.1.48.1";
	policyList[1] = "2.16.840.1.101.3.1.48.2";
	policyList[2] = "2.16.840.1.101.3.1.48.3";
	sigObj.setCertificatePolicy(policyList);

C Interface	ISignature_put_CertificatePolicy(ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_ CertificatePolicy (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim policyList(3) policyList[0] = "2.16.840.1.101.3.1.48.1"; policyList[1] = "2.16.840.1.101.3.1.48.2"; policyList[2] = "2.16.840.1.101.3.1.48.3"; sigObj.CertificatePolicy = policyList Or string [] policyList = new String[3]; policyList[0] = "2.16.840.1.101.3.1.48.1"; policyList[1] = "2.16.840.1.101.3.1.48.2"; policyList[2] = "2.16.840.1.101.3.1.48.3"; sigObj.CertificatePolicy = policyList
Supported Platforms	All

This sets the user constrained policy set for subsequent certificate chain validations.

Parameters:

newVal The given user policy constraints.

CertificatePolicyChecking

Read / Write	Write only.
IDL File Declaration	[propput, id(74), helpstring("property CertificatePolicyChecking")] HRESULT CertificatePolicyChecking([in] BOOL newVal);
Java Interface	public void setCertificatePolicyChecking (int lastParam); sigObj.setCertificatePolicyChecking (1); // Enable Certificate Policy Validation
C Interface	ISignature_put_CertificatePolicyChecking (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_CertificatePolicyChecking (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CertificatePolicyChecking = 1
Supported Platforms	All

This property enables or disables certificate policy validation. The default value is 0 (zero) or disabled. If it is set to a non-zero value, certificate policy validation gets enabled for all calls to SecureXMLVerify() and all signature creation methods.

If DoDCompliance is set to one, any attempt to set this property to zero is ignored.

Parameters:

newVal	Sets the CertificatePolicyChecking property to newVal

CertificatePolicyExplicit

Read / Write	Both Read and Write.
IDL File Declaration	[propget, id(79), helpstring("property CertificatePolicyExplicit")] HRESULT CertificatePolicyExplicit([out, retval] BOOL *pVal); [propput, id(79), helpstring("property CertificatePolicyExplicit")] HRESULT CertificatePolicyExplicit([in] BOOL newVal);

Java Interface	public void setCertificatePolicyExplicit (int lastParam);
	public int getCertificatePolicyExplicit ();
	sigObj.setCertificatePolicyExplicit (1); // Making certificate policy explicit
	int policyExplicitStateVariable = sigObj.getCertificatePolicyExplicit();

C Interface	ISignature_put_CertificatePolicyExplicit (ISignature *pSig, BOOL newVal); ISignature_get_CertificatePolicyExplicit (ISignature *pSig, BOOL *pVal);
C++ Interface	pSig->put_CertificatePolicyExplicit (BOOL newVal); pSig->get_CertificatePolicyExplicit (BOOL *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CertificatePolicyExplicit = 1 int policyExplicitStateVariable = sigObj.CertificatePolicyExplicit
Supported Platforms	All

This property enables or disables certificate policy explicit flag. The default value is 0 (zero) or disabled. If it is set to a non-zero value, certificate policy becomes explicit for all calls to SecureXMLVerify() and all signature creation methods. During a read operation it returns the value of the PolicyExplicit state variable after applying the policy rules to the current certificate chain.

If DoDCompliance is set to one, any attempt to set this property to zero is ignored.

Parameters:

newVal	Sets the CertificatePolicyExplicit property to newVal
pVal	Gets the PolicyExplicit state variable after applying the policy rules to the current certificate chain.

CertificateTrustExplicit

Read / Write	Write only.
IDL File Declaration	[propput, id(80), helpstring("property CertificateTrustExplicit")] HRESULT CertificateTrustExplicit([in] BOOL newVal);
Java Interface	public void setCertificateTrustExplicit (int lastParam); sigObj.setCertificateTrustExplicit (1); // Making certificate policy explicit
C Interface	ISignature_put_CertificateTrustExplicit (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_CertificateTrustExplicit (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CertificateTrustExplicit = 1
Supported Platforms	All

This property is used to restrict the trusted roots to a limited set of certificates specified by the ***TrustedRoots*** property. If CertificateTrustExplicit is set to 1, all certificate validation procedures allow only certificates whose root certificate is one of the certificates contained in ***TrustedRoots***. If ***TrustedRoots*** is empty and CertificateTrustExplicit is enabled, all certificate validation will fail. By default CertificateTrustExplicit is disabled.

If DoDCompliance is set to one, any attempt to set this property to zero is ignored.

Parameters:

newVal Sets the CertificateTrustExplicit property to newVal

CertIssuer

Read / Write	Read Only
IDL File Declaration	[propget, id(31), helpstring("property CertIssuer")] HRESULT CertIssuer([in] BSTR sigId, [out, retval] BSTR *pVal);
Java Interface	public String getCertIssuer(String sigId); String certIssuer = sigObj.getCertIssuer(sigId);
C Interface	ISignature_get_CertIssuer(ISignature *pSig, BSTR sigId, BSTR *pVal);
C++ Interface	pSig->get_CertIssuer(BSTR sigId, BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CertIssuer(sigId)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default). It returns the certificate issuer of the X509Certificate used to create the signature with signature id = sigId.

Parameters:

sigId	This is the signature ID of a signature which was verified previously. A signature ID is obtained by calling SignatureID() property getter.
pVal	On return *pVal is set to the certificate issuing authority's name for the signer's certificate in the given signature.

GetLastError can return the following error codes:

SIG_NOT_FOUND	A signature with matching sigId was not found in the signature just verified.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

CertRevocationDate

Read / Write	Read only.
IDL File Declaration	[propget, id(169), helpstring("property CertRevocationDate")] HRESULT CertRevocationDate([out, retval] BSTR* pVal);
Java Interface	public String getCertRevocationDate () String certRevocationDate = sigObj.getCertRevocationDate (); // Get the certificate revocation date of the last certificate validation performed
C Interface	ISignature_get_CertRevocationDate (ISignature *pSig, BSTR *pVal);
C++ Interface	pSig->get_CertRevocationDate (BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	certRevocationDate = sigObj.CertRevocationDate
Supported Platforms	All

Currently this property is supported only for OCSP based certificate validation. It returns the certificate revocation date of the last OCSP based certificate validation performed.

Parameters:

pVal Returns the certificate revocation date of the last OCSP based certificate validation performed.

CertSerialNumber

Read / Write	Read Only
IDL File Declaration	[propget, id(33), helpstring("property CertSerialNumber")] HRESULT CertSerialNumber([in] BSTR sigId, [out, retval] BSTR *pVal);
Java Interface	public String getCertSerialNumber(String sigId); String certSerialNumber = sigObj.getCertSerialNumber(sigId);
C Interface	ISignature_get_CertSerialNumber(ISignature *pSig, BSTR sigId, BSTR *pVal);
C++ Interface	pSig->get_CertSerialNumber(BSTR sigId, BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CertSerialNumber(sigId)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default). It returns the certificate serial number of the X509Certificate used to create the signature with signature id = sigId.

Parameters:

sigId	This is the signature ID of a signature which was verified previously. A signature ID is obtained by calling SignatureID() property getter.
pVal	On return *pVal is set to the certificate serial number of the signer's certificate in the given signature.

GetLastError can return the following error codes:

SIG_NOT_FOUND	A signature with matching sigId was not found in the signature just verified.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

CertSerialNumberFormat

Read / Write	Both
IDL File Declaration	[propget, id(115), helpstring("property CertSerialNumberFormat")] HRESULT CertSerialNumberFormat([out,retval] LONG *pValretval); [propput, id(115), helpstring("property CertSerialNumberFormat")] HRESULT CertSerialNumberFormat([in] LONG newVal);

Java Interface	public int getCertSerialNumberFormat();
	public void setCertSerialNumberFormat(int lastParam);
	int certSerialNumberFormat = sigObj.getCertSerialNumberFormat();
	sigObj.setCertSerialNumberFormat(1); // Set the certificate serial number format to plain hex

C Interface	ISignature_get_CertSerialNumberFormat(ISignature *pSig, LONG *pVal); ISignature_put_CertSerialNumberFormat(ISignature *pSig, LONG newVal);
C++ Interface	pSig->get_CertSerialNumberFormat(LONG *pVal); pSig->put_CertSerialNumberFormat(newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CertSerialNumberFormat sigObj.CertSerialNumberFormat = 1
Supported Platforms	All

By default SecureXML uses base64 encoded binary values for certificate serial number. If CertSerialNumberFormat is set to 1, it uses plain hex format instead.

Parameters:

pValretval	Returns the current value of CertSerialNumberFormat property.
newVal	Sets the CertSerialNumberFormat to the given value. Curretly the only legal input is 1 and it will switch the certificate serial number format to plain hex.

CertValidationTransactionId

Read / Write	Both
IDL File Declaration	[propget, id(159), helpstring("property CertValidationTransactionId")] HRESULT CertValidationTransactionId([out, retval] BSTR* pVal); [propput, id(159), helpstring("property CertValidationTransactionId")] HRESULT CertValidationTransactionId([in] BSTR newVal);

Java Interface	public String getCertValidationTransactionId();
	public void setCertValidationTransactionId(String lastParam);
	String certValidationTransactionId = sigObj.getCertValidationTransactionId();
	sigObj.setCertValidationTransactionId ("1234"); // Set the certificate validation transaction Id to "1234"

C Interface	ISignature_get_CertValidationTransactionId (ISignature *pSig, BSTR *pVal); ISignature_put_CertValidationTransactionId (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_CertValidationTransactionId (BSTR *pVal); pSig->put_CertValidationTransactionId (newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.CertValidationTransactionId sigObj.CertValidationTransactionId = "1234"
Supported Platforms	All

Gets/sets the certification validation transaction id for the most recent certificate validation request.

If CAM is used for validation the set value has not effect. The read value is the value retured with the CAM validation response.

If OCSP is used for certificate validation, it sets the byte array for OCSP Nonce. Please note the format of this parameter for OCSP certificate validation is assumed to be a series of bytes given as ASCII encoded hex. Each byte is represented by two ASCII characters. For example a valid OCSP Nonce could be "035F34", where it contains three bytes 0x03, 0x5F and 0x34. If the input is provided as "35F34", i.e. the first zero is missing, SecureXML will add this zero in order to make the given Nonce value a valid series of pairs of nibbles for each byte provided.

Parameters:

pVal	Returns the current value of CertValidationTransactionId property.
newVal	Sets the CertValidationTransactionId to the given value.

ConfigFileName

Read / Write	Write only
IDL File Declaration	[propput, id(92), helpstring("property ConfigFileName")] HRESULT ConfigFileName([in] BSTR newVal);
Java Interface	public void setConfigFileName(String newVal); sigObj.setConfigFileName ("C:\\temp\\myconfig.xml"); // Set config file name
C Interface	ISignature_put_ConfigFileName (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_ConfigFileName (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ConfigFileName = "C:\\temp\\myconfig.xml"
Supported Platforms	All

Sets the config file name to newVal. When this property is set, SecureXML reads the given config file and sets appropriate properties to the default values provided in the config file.

Parameters:

newVal	The new value for ConfigFileName, which must be a valid xml file containing ***key = 'ObjectProperty', value='PropertyValue'*** pairs.

CRLCacheDbConnectionString

Read / Write	Both
IDL File Declaration	[propget, id(121), helpstring("property CRLCacheDbConnectionString")] HRESULT CRLCacheDbConnectionString([out, retval] BSTR* pVal); [propput, id(121), helpstring("property CRLCacheDbConnectionString")] HRESULT CRLCacheDbConnectionString([in] BSTR newVal);
Java Interface	public String getCRLCacheDbConnectionString() ; public void setCRLCacheDbConnectionString(String lastParam) ; String curCRLCacheDbConnectionStringVal = sigObj.getCRLCacheDbConnectionString (); String crlConnStr = "driver={sql server};SERVER=(local);Database=SecureXML;UID=sa; PWD=password; Persist Security Info=False;Use Procedure for Prepare=1;Auto Translate=True;Use Encryption for Data=False;Tag with column collation when possible=False;"; sigObj.setCRLCacheDbConnectionString (crlConnStr);
C Interface	ISignature_get_CRLCacheDbConnectionString (ISignature *pSig, BSTR *pVal); ISignature_put_CRLCacheDbConnectionString (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_CRLCacheDbConnectionString (BSTR *pVal); pSig->put_CRLCacheDbConnectionString (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CRLCacheDbConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files\Infomosaic\SecureXML\SecureXML.mdb"; pVal = sigObj.CRLCacheDbConnectionString
Supported Platforms	Windows Only

Set this property to appropriate value before enabling UseCRLCache property. The default value for this property is "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files\Infomosaic\SecureXML\SecureXML.mdb". If you wish to use a different database for CRL Cache, please set this property to point to your database and provider.

The following is the SQL statement for creating the CRLCache table for Microsoft SQL Server:

```
CREATE TABLE [CRLCache] (
        [CertificateIssuer] [nvarchar] (255) DEFAULT (null),
        [CRLData] [ntext] DEFAULT (null),
        [FetchTime] [datetime] DEFAULT (getdate())
        )
```

The connection string for Microsoft SQL Server database is

"driver={sql server};SERVER=(local);Database=SecureXML;UID=sa; PWD=password; Persist Security Info=False;Use Procedure for Prepare=1;Auto Translate=True;Use Encryption for Data=False;Tag with column collation when possible=False;";

Parameters:

newVal	Sets the CRLCacheDbConnectionString to newVal.
pVal	Returns the current settings for CRLCacheDbConnectionString.

CRLCacheTimeoutInMinutes

Read / Write	Write Only
IDL File Declaration	[propput, id(123), helpstring("property CRLCacheTimeoutInMinutes")] HRESULT CRLCacheTimeoutInMinutes([in] ULONG newVal);
Java Interface	public void setCRLCacheTimeoutInMinutes (int lastParam) ; sigObj.setCRLCacheTimeoutInMinutes (120); // Force a CRL fetch if at least 120 minutes have passed since previous fetch.
C Interface	ISignature_put_CRLCacheTimeoutInMinutes (ISignature *pSig, ULONG newVal);
C++ Interface	pSig->put_CRLCacheTimeoutInMinutes (ULONG newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CRLCacheTimeoutInMinutes = 120
Supported Platforms	Windows Only

Normally, if CRLCache is enabled, SecureXML will fetch CRL if the current time is greater than or equal to the NextUpdate time metioned in the CRL. Some Certificate Authorities may update the CRLs sooner than the next update time if there was indeed a certificate revocation. In order to force SecureXML to get the new CRL before the next update time, you can set CRLCacheTimeoutInMinutes property to a non-zero value. For example if CRLCacheTimeoutInMinutes is set to 120, SecureXML will fetch a new CRL for the certificate being verified if the CRL fetch time for the CRL in the cache is 120 or more minutes old. Please note that if the next update time is less than the CRLCacheTimeoutInMinutes setting then next update time is used to fetch the CRL and CRLCacheTimeoutInMinutes has no effect.

Parameters:

newVal Sets the CRLCacheTimeoutInMinutes to newVal.

CrlChecking

Read / Write	Read/Write
IDL File Declaration	[propget, id(64), helpstring("property CrlChecking")] HRESULT CrlChecking([out, retval] BOOL *pVal); [propput, id(64), helpstring("property CrlChecking")] HRESULT CrlChecking([in] BOOL newVal);

Java Interface	public int getCrlChecking();
	public void setCrlChecking(int lastParam);
	int curCrlCheckingVal = sigObj.getCrlChecking();
	sigObj.setCrlChecking(1); // Enable CRL Checking

C Interface	ISignature_get_CrlChecking (ISignature *pSig, BOOL *pVal); ISignature_put_CrlChecking (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_CrlChecking (BOOL *pVal); pSig->put_CrlChecking (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.CrlChecking = 1 pVal = sigObj.CrlChecking
Supported Platforms	All

This property enables or disables certificate verification against its certificate revocation list. The default value is 0 (zero) or disabled. If it is set to a non-zero value, CRL checking gets enabled for all calls to SecureXMLVerify() method. If the signature is valid but the certificate is revoked, the last error element of the SecureXMLResponse XML contains value 61 or CERT_TRUST_ERROR.

It is recommended that if CRL checking be enabled, TimeStamping should also be enabled for signature creation. This allows the CRL checking to be done for the time when the signature was created. If there is no TimeStamp information in the signature being verified and CRL checking is enabled, the current system time is used for certificate revocation verification.

If DoDCompliance is set to one, any attempt to set this property to zero is ignored.

Parameters:

newVal Sets the CrlChecking property to newVal

pVal On return *pVal is set to the current setting for this property.

GetLastError can return the following error codes:

None.

CRLLocation

Read / Write	Write Only
IDL File Declaration	[propput, id(69), helpstring("property CRLLocation")] HRESULT CRLLocation([in] VARIANT newVal);
Java Interface	public void setCRLLocation (String [] lastParam); String [] crlList = new String[3]; crlList[0] = "c:\mycrlLocation\crl1.crl"; crlList[1] = "http://www.infomosaic.com/Crlfile.crl"; crlList[2] = "ldap://myldapserver.ldap.com/crl.crl"; sigObj.setCRLLocation (crlList);
C Interface	ISignature_put_CRLLocation (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_CRLLocation (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim crlList(3) crlList[0] = "c:\mycrlLocation\crl1.crl"; crlList[1] = "http://www.infomosaic.com/Crlfile.crl"; crlList[2] = "ldap://myldapserver.ldap.com/crl.crl"; sigObj.CRLLocation = crlList Or string [] crlList = new String[3]; crlList[0] = "c:\mycrlLocation\crl1.crl"; crlList[1] = "http://www.infomosaic.com/Crlfile.crl"; crlList[2] = "ldap://myldapserver.ldap.com/crl.crl"; sigObj.CRLLocation = crlList
Supported Platforms	All

If the certificates do not have a distribution point specified then SecureXML will try to use the CRL Locations specified by this property. You must have CrlChecking or DoDCompliance set to 1 for CRL to be verified.

Parameters:

newVal The given CRL files.

DecryptionPFXCertFile

Read / Write	Write Only
IDL File Declaration	[propput, id(102), helpstring("property DecryptionPFXCertFile")] HRESULT DecryptionPFXCertFile([in] BSTR newVal);
Java Interface	public void setDecryptionPFXCertFile(String lastParam); sigObj.setDecryptionPFXCertFile ("C:\\temp\\mycert.pfx");
C Interface	ISignature_put_DecryptionPFXCertFile (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_DecryptionPFXCertFile (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.DecryptionPFXCertFile = "C:\temp\mycert.pfx"
Supported Platforms	All

You can set this property to PFX or P12 file certificate containing the private key for decrypting a message encrypted using the corresponding public key. You must also set DecryptUsingPFXFileCert property to 1 and set DecryptionPFXPassword to the password used for protecting the private key in order to successfully decrypt the encrypted message by calling DecryptStr or DecryptFile methods.

Parameters:

newVal	The URI pointing to the PFX or P12 file to be used for decrypting in subsequent calls to DecryptStr and DecryptFile.

DecryptionPFXPassword

Read / Write	Write Only
IDL File Declaration	[propput, id(104), helpstring("property DecryptionPFXPassword")] HRESULT DecryptionPFXPassword([in] BSTR newVal);
Java Interface	public void setDecryptionPFXPassword(String lastParam); sigObj.setDecryptionPFXPassword ("password");
C Interface	ISignature_put_DecryptionPFXPassword (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_DecryptionPFXPassword (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.DecryptionPFXPassword = "password"
Supported Platforms	All

Set this property to the password used for protecting the private key for the PFX file specified by DecryptionPFXCertFile property in order to successfully decrypt the encrypted message by calling DecryptStr or DecryptFile methods. You must also set DecryptUsingPFXFileCert property to 1.

Parameters:

newVal	The password to be used for accessing the private key contained in the PFX file pointed to by DecryptionPFXCertFile property during a call to DecryptStr and DecryptFile methods.

DecryptUsingPFXFileCert

Read / Write	Write Only
IDL File Declaration	[propput, id(103), helpstring("property DecryptUsingPFXFileCert")] HRESULT DecryptUsingPFXFileCert([in] BOOL newVal);
Java Interface	public void setDecryptUsingPFXFileCert (int lastParam); sigObj.setDecryptUsingPFXFileCert (1);
C Interface	ISignature_put_DecryptUsingPFXFileCert (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_DecryptUsingPFXFileCert (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.DecryptUsingPFXFileCert = 1
Supported Platforms	All

You must set DecryptUsingPFXFileCert property to 1 in order for SecureXML to look at the DecryptionPFXCertFile and DecryptionPFXPassword properties during a call to DecryptStr and DecryptFile methods. When all three of these properties are set correctly SecureXML will use the PFX file certificate to decrypt the encrypted message instead of looking for a certificate in the current user's "MY" store or in the store set by SetStoreName method.

Parameters:

newVal 1 to enable PFX file based decryption, 0 (default) to disable it.

DetachedObjects

Read / Write	Write Only
IDL File Declaration	[propput, id(99), helpstring("property DetachedObjects")] HRESULT DetachedObjects([in] VARIANT newVal);

Java Interface	public void setDetachedObjects(String [] lastParam);
	String [] detachedObjectList = new String[3];
	detachedObjectList[0] = "http://www.infomosaic.com/index.html";
	detachedObjectList[1] = "file:///c:/temp/mydata.doc";
	detachedObjectList[2] = "G:\temp\mySpreadSheet.xls";
	sigObj.setDetachedObjects(detachedObjectList);

C Interface	ISignature_put_DetachedObjects(ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_DetachedObjects (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim detachedObjectList(3) detachedObjectList[0] = "http://www.infomosaic.com/index.html"; detachedObjectList[1] = "file:///c:/temp/mydata.doc"; detachedObjectList[2] = "G:\temp\mySpreadSheet.xls"; sigObj.DetachedObjects = DetachedObjectList Or string [] detachedObjectList = new String[3]; detachedObjectList[0] = "http://www.infomosaic.com/index.html"; detachedObjectList[1] = "file:///c:/temp/mydata.doc"; detachedObjectList[2] = "G:\temp\mySpreadSheet.xls"; sigObj.DetachedObjects = DetachedObjectList
Supported Platforms	All

If DetachedObjects is non-null, all the objects pointed to by DetachedObjects are included inside the current signature (as in an enveloping signature) as external object references. If any of these objects are not accessible, the signature creation will fail.

Parameters:

newVal The given objects to be included in a detached manner for the current signature.

DetailedVerificationFlag

Read / Write	Write Only
IDL File Declaration	[propput, id(85), helpstring("property DetailedVerificationFlag")] HRESULT DetailedVerificationFlag([in] BOOL newVal);
Java Interface	public void setDetailedVerificationFlag(int lastParam); sigObj.setDetailedVerificationFlag(0); // Disable detailed verification
C Interface	ISignature_get_DetailedVerificationFlag (ISignature *pSig, BOOL *pVal); ISignature_put_DetailedVerificationFlag (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_DetailedVerificationFlag (BOOL *pVal); pSig->put_DetailedVerificationFlag (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.DetailedVerificationFlag = 0 pVal = sigObj.DetailedVerificationFlag
Supported Platforms	All

When DetailedVerificationFlag is set to zero (the default is 1), the signature verification methods do not extract and save the signed references to disk as temporary files from the signed XML. It makes the signature verification disk I/O free and hence faster. The signed objects are available in memory buffer though and can be accessed via appropriate APIs. If the signed documents (i.e. references) are needed as temporary files stored on the local drive, set the DetailedVerificationFlag back to 1 and verify signature again.

Setting DetailedVerificationFlag to 2, does not prepare signed objects for access after signature verification. This reduces the total memory requirement during signature verification by more than 50% and makes verification 20% faster. This feature is most useful when a signature produced over large detached signed data is being verified.

It is recommended that if you are using SecureXMLVerify () / SecureXMLVerifyByteArray () methods for verfying signature, you set DetailedVerificationFlag to zero since you will always get the detailed information in the response. This flag becomes useful for SecureXMLVerify() method if you have a detached reference which could not be resolved during verification and you need to specify the failed URI's object locations by setting FailedUriFullPath object property.

Parameters:

newVal Sets the DetailedVerificationFlag property to newVal

DigestObjectStatus

Read / Write	Read Only
IDL File Declaration	[propget, id(9), helpstring("property DigestObjectStatus")] HRESULT DigestObjectStatus([in] long sigIndex, [in] long uriIndex, [out, retval] BOOL *pVal);
Java Interface	public int getDigestObjectStatus(int sigIndex, int uriIndex); int digestObjStatus = sigObj.getDigestObjectStatus(sigIndex, uriIndex);
C Interface	ISignature_get_DigestObjectStatus(ISignature *pSig, long sigIndex, long uriIndex, BOOL *pVal);
C++ Interface	pSig->get_DigestObjectStatus(long sigIndex, long uriIndex, BOOL *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.DigestObjectStatus(sigIndex, uriIndex)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default).

Parameters:

sigIndex	This is the signature index of a signature which was verified previously. A signature index can be from 0 to (SignatureCount – 1).
uriIndex	This is the index number for the signed URI in the signature corresponding to sigIndex. A uriIndex can be from 0 to TotalUriCount().
pVal	On return *pVal is True if the digest verification for the given URI was successful. It is set to False if the digest verification fails.

GetLastError can return the following error codes:

SIG_INDEX_ERROR	sigIndex provided is >= SignatureCount or < 0.
URI_INDEX_ERROR	uriIndex provided is >= maxURI where maxURI is the number of URI referred to by the current signature
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

DocumentURI

Read / Write	Read Only
IDL File Declaration	[propget, id(8), helpstring("property DocumentURI")] HRESULT DocumentURI([in] long sigIndex, [in] long uriIndex, [out, retval] BSTR *pVal);
Java Interface	public String getDocumentURI(int sigIndex, int uriIndex); String digestAlgo = sigObj.getDocumentURI(sigIndex, uriIndex);
C Interface	ISignature_get_DocumentURI(ISignature *pSig, long sigIndex, long uriIndex, BSTR *pVal);
C++ Interface	pSig->get_DocumentURI(long sigIndex, long uriIndex, BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.DocumentURI(sigIndex, uriIndex)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default).

Parameters:

sigIndex	This is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount -1.
uriIndex	This is the index number for the signed URI in the signature corresponding to sigIndex. A uriIndex can be from 0 to TotalUriCount().
pVal	On return *pVal is set to the URI that was referred to during signature creation.

GetLastError can return the following error codes:

SIG_INDEX_ERROR	sigIndex provided is >= SignatureCount or < 0.
URI_INDEX_ERROR	uriIndex provided is >= maxURI where maxURI is the number of URI referred to by the current signature
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

DoDCompliance

Read / Write	Write only.
IDL File Declaration	[propput, id(78), helpstring("property DoDCompliance")] HRESULT DoDCompliance([in] BOOL newVal);
Java Interface	public void setDoDCompliance (int lastParam); sigObj.setDoDCompliance (1); // Enable DoDCompliance
C Interface	ISignature_put_DoDCompliance (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_DoDCompliance (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.DoDCompliance = 1
Supported Platforms	All

This property sets and enables the following properties for the current SecureXML object:

- CertificatePolicyChecking
- CertificateChainValidation
- CertificatePathLengthChecking
- CrlChecking
- CertificateTrustExplicit

If newVal is zero, it disabled all of the above validation flags.
Once DoDCompliance is set to 1 (i.e. enabled) the above object properties can no longer be set/reset by invoking corresponding property getter/setter API. DoDCompliance must be set to zero before any attempt to change the above individual properties can take effect.

Parameters:

newVal	Sets the DoDCompliance property to newVal

EnvelopingFlag

Read / Write	Both
IDL File Declaration	[propget, id(1), helpstring("property EnvelopingFlag")] HRESULT EnvelopingFlag([out, retval] short *pVal); [propput, id(1), helpstring("property EnvelopingFlag")] HRESULT EnvelopingFlag([in] short newVal);

Java Interface	public short getEnvelopingFlag();
	public void setEnvelopingFlag(short lastParam);
	short curEnvelopingFlagVal = sigObj.getEnvelopingFlag();
	sigObj.setEnvelopingFlag(2); // Set enveloping flag to Enveloped

C Interface	ISignature_get_EnvelopingFlag (ISignature *pSig, short *pVal); ISignature_put_EnvelopingFlag (ISignature *pSig, short newVal);
C++ Interface	pSig->get_EnvelopingFlag (short *pVal); pSig->put_EnvelopingFlag (short newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.EnvelopingFlag = 2 pVal = sigObj.EnvelopingFlag
Supported Platforms	All

This must be set before the Sign() method is invoked.

Parameters:

newVal Sets the Enveloping flag to newVal.
The following are the allowed values and their meaning:

Value	Signature Type	Allowed URI for the Sign Method
0	Detached Signature	Can be file or URL
1	Enveloping	Can be file or URL
2	Enveloped	Must be a file. URL not allowed

pVal Returns the current settings of the EnvelopingFlag.

ExcludeSignerCertificate

Read / Write	Write only.
IDL File Declaration	[propput, id(88), helpstring("property ExcludeSignerCertificate")] HRESULT ExcludeSignerCertificate([in] BOOL newVal);
Java Interface	public void setExcludeSignerCertificate(int newVal) sigObj.setExcludeSignerCertificate (1); // Don't include signer certificate
C Interface	ISignature_put_ExcludeSignerCertificate (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_ExcludeSignerCertificate (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ExcludeSignerCertificate = 2; //Include only the end certificate and not the whole certificate chain
Supported Platforms	All

By default, SecureXML includes a copy of the signer's X509 certificate chain in every XML signature created. This increases the size of the signed XML by approximately 4-6 Kbytes in most cases. If the signature being created is going to be consumed by internal business users, it might be more space efficient to store a copy of the X509 certificate in a certral database and not include the X509 data with each signature. The signed XML will still contain the signer's public key. During signature verification, if detailed signer indentity information is needed one can query the central database for the certificate corresponding to the signer public key. Setting ExcludeSignerCertificate to 1, makes SecureXML not include the signer's X509 certificate with each signature created. Setting ExcludeSignerCertificate to 2 makes SecureXML include only the end user certificate and not the whole certificate chain in the signature being created.

Parameters:

newVal Sets the ExcludeSignerCertificate property to newVal.

HMACPassword

Read / Write	Both
IDL File Declaration	[propget, id(83), helpstring("property HMACPassword")] HRESULT HMACPassword([out, retval] BSTR *pVal); [propput, id(83), helpstring("property HMACPassword")] HRESULT HMACPassword([in] BSTR newVal);

Java Interface	public String getHMACPassword();
	public void setHMACPassword(String lastParam);
	String curHMACPasswordVal = sigObj.getHMACPassword();
	sigObj.setHMACPassword("password"); // Set HMACPassword to 'password'

C Interface	ISignature_get_HMACPassword (ISignature *pSig, BSTR *pVal); ISignature_put_HMACPassword (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_HMACPassword (BSTR *pVal); pSig->put_HMACPassword (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.HMACPassword = "password" pVal = sigObj.HMACPassword
Supported Platforms	All

If UseHMAC property is set, the value of HMACPassword is used for creating XML Signature. It is also used for signature verification by all signature verification methods. If HMACPassword is NULL or not set, and UseHMAC property is set to 1, signature creation fails.

Parameters:

newVal	Sets the HMAC password to newVal.
pVal	Returns the current settings for HMAC password.

FloatingLicense

Read / Write	Both
IDL File Declaration	[propget, id(143), helpstring("property FloatingLicense")] HRESULT FloatingLicense([out,retval] BSTR *pVal); [propput, id(143), helpstring("property FloatingLicense")] HRESULT FloatingLicense([in] BSTR newVal);
Java Interface	public String getFloatingLicense (); public void setFloatingLicense (String lastParam); String floatingLicense = sigObj.getFloatingLicense (); sigObj.setFloatingLicense (floatingLicense);
C Interface	ISignature_get_FloatingLicense (ISignature *pSig, BSTR *pVal); ISignature_put_FloatingLicense (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_FloatingLicense (BSTR *pVal); pSig->put_FloatingLicense (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.FloatingLicense = floatingLicense pVal = sigObj.FloatingLicense
Supported Platforms	All

Use the property getter on a licensed server. Bring the returned value to the web client; use this value to set FloatingLicense property on the client before invoking any signature creation operation. Please note that the web client must be running on a non-server operating system for the floating licensing to work. Please do not use the Signature object used on the server for your application for obtaining the floating license as it may interfier with your application. Instead, you shouls create a new Signature object, call the FloatingLicense property getter, save the returned value for sending to the client side and then destroy this Signature object (by calling destroy() if using Java or by setting the variable holding this object to Nothing or null if using other languages).

It is violation of SecureXML End User License Agreement to apply the floating license value returned from a licensed server to a client not in direct communication with the licensed server running the application for which a development license was obtained.

Parameters:

newVal	Sets the FloatingLicense to newVal.
pVal	Returns a new FloatingLicense value which can be applied to a client instance of SecureXML for enabling signature creation on the client.

IgnoreIncompleteSignature

Read / Write	Read/Write
IDL File Declaration	[propget, id(160), helpstring("property IgnoreIncompleteSignature")] HRESULT IgnoreIncompleteSignature([out, retval] BOOL* pVal); [propput, id(160), helpstring("property IgnoreIncompleteSignature")] HRESULT IgnoreIncompleteSignature([in] BOOL newVal);

Java Interface	public int getIgnoreIncompleteSignature ();
	public void setIgnoreIncompleteSignature (int lastParam);
	int curIgnoreIncompleteSignature = sigObj.getIgnoreIncompleteSignature ();
	sigObj.setIgnoreIncompleteSignature (1); // Make SecureXML ignore incomplete signature
	// elements during signature verification

C Interface	ISignature_get_IgnoreIncompleteSignature (ISignature *pSig, BOOL *pVal); ISignature_put_IgnoreIncompleteSignature (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_IgnoreIncompleteSignature (BOOL *pVal); pSig->put_IgnoreIncompleteSignature (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.IgnoreIncompleteSignature = 1 pVal = sigObj.IgnoreIncompleteSignature
Supported Platforms	All

By default SecureXML attempts to verify each and every signature element found in the input document during any of the signature verification methods. If IgnoreIncompleteSignature is set to 1, all signature verification methods ignore all signature elements which do not have a populated <SignatureValue> subelement. The default value is 0 or disabled.

Parameters:

pVal	Returns the current setting for the IgnoreIncompleteSignature flag.
newVal	Sets the IgnoreIncompleteSignature flag.

IncludeCamResponse

Read / Write	Read/Write
IDL File Declaration	[propget, id(16), helpstring("property IncludeCamResponse")] HRESULT IncludeCamResponse([out, retval] BOOL *pVal); [propput, id(16), helpstring("property IncludeCamResponse")] HRESULT IncludeCamResponse([in] BOOL newVal);

Java Interface	public int getIncludeCamResponse ();
	public void setIncludeCamResponse (int lastParam);
	int curIncludeCamResponseVal = sigObj.getIncludeCamResponse ();
	sigObj.setIncludeCamResponse (1); // Enable CAM certificate validation response inclusion

C Interface	ISignature_get_IncludeCamResponse (ISignature *pSig, BOOL *pVal); ISignature_put_IncludeCamResponse (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_IncludeCamResponse (BOOL *pVal); pSig->put_IncludeCamResponse (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.IncludeCamResponse = 1 pVal = sigObj.IncludeCamResponse
Supported Platforms	All

Enables or disables CAM certificate validation inclusion with the current signature being created. If it is enabled, a signature property called "CamValidationResponse" is added for the current signature being created. In addition to setting IncludeCamResponse, the application must also set proper CamServerHost, CamServerPort parameters and set UseCam to 1 for IncludeCamResponse to be functional.

Parameters:

pVal	Returns the current setting for the IncludeCamResponse flag.
newVal	Sets the IncludeCamResponse flag. A zero value disables inclusion of the CAM certificate validation response and 1 value enables it.

IncludeCRLInSignature

Read / Write	Write only
IDL File Declaration	[propput, id(145), helpstring("property IncludeCRLInSignature")] HRESULT IncludeCRLInSignature([in] BOOL newVal);
Java Interface	public void setIncludeCRLInSignature (int newVal); sigObj.setIncludeCRLInSignature (1); // Enable CRL inclusion
C Interface	ISignature_put_IncludeCRLInSignature (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_IncludeCRLInSignature (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.IncludeCRLInSignature = 1
Supported Platforms	All

Enables or disables CRL data inclusion with the current signature being created. The default value is 0 (disabled). In addition to setting IncludeCRLInSignature, the application must also set CrlChecking to 1 for IncludeCRLInSignature to be functional. This will help validate the certificate offline provide TimeStamping is also set to 1. During signature verification, if CrlChecking is disabled and if the signature being verified contains the CRL (i.e. IncludeCRLInSignature was enabled during signature creation), SecureXML verifies the certificate with the included CRL data. If CrlChecking is enabled during signature verification, live CRL is fetched and the included CRL data, if any, is ignored.

Parameters:

newVal	Sets the IncludeCRLInSignature flag. A zero value (default) disables inclusion of the CRL data and 1 value enables it.

IncludeOcspResponse

Read / Write	Read/Write
IDL File Declaration	[propget, id(170), helpstring("property IncludeOcspResponse")] HRESULT IncludeOcspResponse([out, retval] LONG* pVal); [propput, id(170), helpstring("property IncludeOcspResponse")] HRESULT IncludeOcspResponse([in] LONG newVal);

Java Interface	public int getIncludeOcspResponse ();
	public void setIncludeOcspResponse (int lastParam);
	int curIncludeOcspResponseVal = sigObj.getIncludeOcspResponse ();
	sigObj.setIncludeOcspResponse (1); // Enable Text OCSP certificate validation response inclusion

C Interface	ISignature_get_IncludeOcspResponse (ISignature *pSig, LONG *pVal); ISignature_put_IncludeOcspResponse (ISignature *pSig, LONG newVal);
C++ Interface	pSig->get_IncludeOcspResponse (LONG *pVal); pSig->put_IncludeOcspResponse (LONG newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.IncludeOcspResponse = 1 pVal = sigObj.IncludeOcspResponse
Supported Platforms	All

Enables or disables OCSP certificate validation response inclusion with the current signature being created. If it is zero (default) the OCSP validation response is not included in the current signature.

If it is enabled, i.e. IncludeOcspResponse is set to 1(include the text response), 2(include the binary response) or 3(include both the text and the binary responses), signature properties called “OcspTextResponse” and/or “OcspBinaryResponse” are added for the current signature being created.

Parameters:

pVal	Returns the current setting for the IncludeOcspResponse flag.
newVal	Sets the IncludeOcspResponse property. A zero value disables inclusion of the OCSP certificate validation response and 1, 2 or 3 values enable it.

InclusiveNamespacePrefixList

Read / Write	Write Only
IDL File Declaration	[propput, id(175), helpstring("property InclusiveNamespacePrefixList")] HRESULT InclusiveNamespacePrefixList([in] VARIANT newVal);
Java Interface	public void setInclusiveNamespacePrefixList (String [] lastParam); String []InclusiveNsPrefixList = new String[3]; InclusiveNsPrefixList [0] = "infomosaic"; InclusiveNsPrefixList [1] = "mynamespace1"; InclusiveNsPrefixList [2] = "mynamespace2"; sigObj.setInclusiveNamespacePrefixList (InclusiveNsPrefixList);
C Interface	ISignature_put_InclusiveNamespacePrefixList (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_InclusiveNamespacePrefixList (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim InclusiveNsPrefixList (3) InclusiveNsPrefixList [0] = "infomosaic"; InclusiveNsPrefixList [1] = "mynamespace1"; InclusiveNsPrefixList [2] = "mynamespace2"; sigObj.InclusiveNamespacePrefixList = InclusiveNsPrefixList Or string [] InclusiveNsPrefixList = new String[3]; InclusiveNsPrefixList [0] = "infomosaic"; InclusiveNsPrefixList [1] = "mynamespace1"; InclusiveNsPrefixList [2] = "mynamespace2"; sigObj.InclusiveNamespacePrefixList = InclusiveNsPrefixList
Supported Platforms	All

For all signature creations where SecureXML adds a reference to the <Signature> element being processed, if the CanonicalizationMethod is either 2 or 3, it adds the namespace prefixs provided by InclusiveNamespacePrefixList property to the inclusive list provide for the exclusive canonicalization transformation for that reference. By default all namespaces (expect the default namespace of the xml-dsig) are excluded during digest calculation.

Parameters:

newVal The given namespace prefixes to include during reference canonicalization.

Language

Read / Write	Both
IDL File Declaration	[propget, id(10), helpstring("property Language")] HRESULT Language([out, retval] BSTR *pVal); [propput, id(10), helpstring("property Language")] HRESULT Language([in] BSTR newVal);
Java Interface	public String getLanguage (); public void setLanguage (String lastParam); String language = sigObj.getLanguage (); sigObj.setLanguage ("JP");
C Interface	ISignature_get_Language (ISignature *pSig, BSTR *pVal); ISignature_put_Language (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_Language (BSTR *pVal); pSig->put_Language (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.Language = "JP" pVal = sigObj.Language
Supported Platforms	Windows Only

This property sets the language for the certificate selection dialog box to be displayed whenever a user certificate selection dialog box is displayed. Approriate dialog resource file must be present in the PATH in order for SecureXML to be able to load the language specific resource dll. The following are the supported values for the input parameter:

1. "JP" for Japanese (resource only dll file name = JapaneseDiag.dll)
2. "EN" for English (default) (resource only dll file name = EnglishDiag.dll)
3. "FR" for French (resource only dll file name = FrenchDiag.dll)
4. "HU" for Hungarian (resource only dll file name = HungarianDiag.dll)
5. "ES" for Spanish (resource only dll file name = SpanishDiag.dll)
6. "DE" for German (resource only dll file name = GermanDiag.dll)
7. "PT" for Portuguese (resource only dll file name = PortugueseDiag.dll)

LicensedUserCount

Read / Write	Read only
IDL File Declaration	[propget, id(144), helpstring("property LicensedUserCount")] HRESULT LicensedUserCount([out, retval] LONG* pVal);
Java Interface	public int getLicensedUserCount (); int licensedUserCount = sigObj.getLicensedUserCount();
C Interface	ISignature_get_LicensedUserCount (ISignature *pSig, LONG *pVal);
C++ Interface	pSig->get_LicensedUserCount (LONG *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	licensedUserCount = sigObj.LicensedUserCount
Supported Platforms	All

It returns the number of users allowed to use the server side signature creation process of SecureXML. All applications using server side signature creation must limit the number of users to the LicensedUserCount in order to comply with SecureXML Server licensing.

Parameters:

pVal	Gets the LicensedUserCount value.

NetscapeStorePassword

Read / Write	Both
IDL File Declaration	[propget, id(125), helpstring("property NetscapeStorePassword")] HRESULT NetscapeStorePassword([out, retval] BSTR* pVal); [propput, id(125), helpstring("property NetscapeStorePassword")] HRESULT NetscapeStorePassword([in] BSTR newVal);
Java Interface	public String getNetscapeStorePassword (); public void setNetscapeStorePassword (String lastParam); String curNetscapeStorePasswordVal = sigObj.getNetscapeStorePassword (); sigObj.setNetscapeStorePassword (“password”); // Set NetscapeStorePassword to “password”
C Interface	ISignature_get_NetscapeStorePassword (ISignature *pSig, BSTR *pVal); ISignature_put_NetscapeStorePassword (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_NetscapeStorePassword (BSTR *pVal); pSig->put_NetscapeStorePassword (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.NetscapeStorePassword = “password” pVal = sigObj.NetscapeStorePassword
Supported Platforms	All

If the user has protected the netscape certificate store with a password, this property must be set to this password before invoking the SetStoreName(“Netscape”) method.

Parameters:

newVal	Sets the NetscapeStorePassword to newVal.
pVal	Returns the current settings for NetscapeStorePassword.

OcspB64Response

Read / Write	Read only.
IDL File Declaration	[propget, id(168), helpstring("property OcspB64Response")] HRESULT OcspB64Response([out, retval] BSTR* pVal);
Java Interface	public String getOcspB64Response() String ocspB64Response = sigObj.getOcspB64Response(); // Get last Ocsp certificate validation binary response as a base64 encoded string
C Interface	ISignature_get_OcspB64Response (ISignature *pSig, BSTR *pVal);
C++ Interface	pSig->get_OcspB64Response (BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	ocspB64Response = sigObj.OcspB64Response
Supported Platforms	All

It returns the last OCSP based certificate validation binary response as a base64 encoded string.

Parameters:

pVal Returns the base 64 encoded OCSP binary response.

OcspReqSignerPFXCertPassword

Read / Write	Both
IDL File Declaration	[propget, id(165), helpstring("property OcspReqSignerPFXCertPassword")] HRESULT OcspReqSignerPFXCertPassword([out, retval] BSTR* pVal); [propput, id(165), helpstring("property OcspReqSignerPFXCertPassword")] HRESULT OcspReqSignerPFXCertPassword([in] BSTR newVal);
Java Interface	public String getOcspReqSignerPFXCertPassword (); public void setOcspReqSignerPFXCertPassword (String lastParam); String curOcspReqSignerPFXCertPasswordVal = sigObj.getOcspReqSignerPFXCertPassword (); sigObj.setOcspReqSignerPFXCertPassword ("mypassword"); // Set password to mypassword
C Interface	ISignature_get_OcspReqSignerPFXCertPassword (ISignature *pSig, BSTR *pVal); ISignature_put_OcspReqSignerPFXCertPassword (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_OcspReqSignerPFXCertPassword (BSTR *pVal); pSig->put_OcspReqSignerPFXCertPassword (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.OcspReqSignerPFXCertPassword = "mypassword" pVal = sigObj.OcspReqSignerPFXCertPassword
Supported Platforms	All

If a signed OCSP request needs to be sent, a valid PFX/P12 file containing the certificate and private key to be used for signing must be provided via the ***OcspReqSignerPFXCertPath*** property. The OcspReqSignerPFXCertPassword property sets the password that needs to be used for accessing the private key contained in the given PFX/P12 file.

Parameters:

newVal	Sets the OcspReqSignerPFXCertPassword to newVal.
pVal	Returns the current value of OcspReqSignerPFXCertPassword

OcspReqSignerPFXCertPath

Read / Write	Both
IDL File Declaration	[propget, id(164), helpstring("property OcspReqSignerPFXCertPath")] HRESULT OcspReqSignerPFXCertPath([out, retval] BSTR* pVal); [propput, id(164), helpstring("property OcspReqSignerPFXCertPath")] HRESULT OcspReqSignerPFXCertPath([in] BSTR newVal);
Java Interface	public String getOcspReqSignerPFXCertPath (); public void setOcspReqSignerPFXCertPath (String lastParam); String curOcspReqSignerPFXCertPathVal = sigObj.getOcspReqSignerPFXCertPath (); sigObj.setOcspReqSignerPFXCertPath ("C:\temp\ocspSigner.pfx");
C Interface	ISignature_get_OcspReqSignerPFXCertPath (ISignature *pSig, BSTR *pVal); ISignature_put_OcspReqSignerPFXCertPath (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_OcspReqSignerPFXCertPath (BSTR *pVal); pSig->put_OcspReqSignerPFXCertPath (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.OcspReqSignerPFXCertPath = "C:\temp\ocspSigner.pfx" pVal = sigObj.OcspReqSignerPFXCertPath
Supported Platforms	All

If a signed OCSP request needs to be sent, a valid PFX/P12 file containing the certificate and private key to be used for signing must be provided via the ***OcspReqSignerPFXCertPath*** property. The ***OcspReqSignerPFXCertPassword*** property must also be set to the password that needs to be used for accessing the private key contained in the given PFX/P12 file.

Parameters:

newVal	Sets the OcspReqSignerPFXCertPath to newVal.
pVal	Returns the current value of OcspReqSignerPFXCertPath

OcspResponderURL

Read / Write	Both
IDL File Declaration	[propget, id(166), helpstring("property OcspResponderURL")] HRESULT OcspResponderURL([out, retval] BSTR* pVal); [propput, id(166), helpstring("property OcspResponderURL")] HRESULT OcspResponderURL([in] BSTR newVal);
Java Interface	public String getOcspResponderURL (); public void setOcspResponderURL (String lastParam); String curOcspResponderURLVal = sigObj.getOcspResponderURL (); sigObj.setOcspResponderURL ("http://www.myocspserver.com");
C Interface	ISignature_get_OcspResponderURL (ISignature *pSig, BSTR *pVal); ISignature_put_OcspResponderURL (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_OcspResponderURL (BSTR *pVal); pSig->put_OcspResponderURL (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.OcspResponderURL = "http://www.myocspserver.com" pVal = sigObj.OcspResponderURL
Supported Platforms	All

By default SecureXML uses the OCSP responder location provided in the AIA extension of the certificate being validated. If a different OCSP responder should be used, the application can set that using the OcspResponderURL property. During a read operation OcspResponderURL returns the actual OCSP responder URL used during the last certificate validation.

Parameters:

newVal	Sets the OcspResponderURL to newVal.
pVal	Returns the value of OcspResponderURL used during the last certificate validation

OcspTextResponse

Read / Write	Read only.
IDL File Declaration	[propget, id(167), helpstring("property OcspTextResponse")] HRESULT OcspTextResponse([out, retval] BSTR* pVal);
Java Interface	public String getOcspTextResponse() String ocspTextResponse = sigObj.getOcspTextResponse(); // Get last Ocsp certificate validation response as a formatted ASCII string
C Interface	ISignature_get_OcspTextResponse (ISignature *pSig, BSTR *pVal);
C++ Interface	pSig->get_OcspTextResponse (BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	ocspTextResponse = sigObj.OcspTextResponse
Supported Platforms	All

It returns the last OCSP based certificate validation response as a formatted ASCII string.

Parameters:

pVal	Returns the formatted ASCII OCSP response.

OcspTrustedRespSignerCertPath

Read / Write	Both
IDL File Declaration	[propget, id(163), helpstring("property OcspTrustedRespSignerCertPath")] HRESULT OcspTrustedRespSignerCertPath([out, retval] BSTR* pVal); [propput, id(163), helpstring("property OcspTrustedRespSignerCertPath")] HRESULT OcspTrustedRespSignerCertPath([in] BSTR newVal);
Java Interface	public String getOcspTrustedRespSignerCertPath (); public void setOcspTrustedRespSignerCertPath (String lastParam); String curOcspTrustedRespSignerCertPathVal = sigObj.getOcspTrustedRespSignerCertPath (); sigObj.setOcspTrustedRespSignerCertPath ("C:\temp\mycertca.pem");
C Interface	ISignature_get_OcspTrustedRespSignerCertPath (ISignature *pSig, BSTR *pVal); ISignature_put_OcspTrustedRespSignerCertPath (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_OcspTrustedRespSignerCertPath (BSTR *pVal); pSig->put_OcspTrustedRespSignerCertPath (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.OcspTrustedRespSignerCertPath = "C:\temp\mycertca.pem" pVal = sigObj.OcspTrustedRespSignerCertPath
Supported Platforms	All

If the application must enforce the OCSP response to be from a known response signer, the signer certificate must be set using the OcspTrustedRespSignerCertPath property. The file pointed to by the OcspTrustedRespSignerCertPath must contain the trusted response signer certificate in PEM format (which is a concatenation of base64 encoded DER certificates separated by -----BEGIN CERTIFICATE---- and -----END CERTIFICATE----- lines.

If OcspTrustedRespSignerCertPath is not set, all responses are considered trusted.

Parameters:

newVal	Sets the OcspTrustedRespSignerCertPath to newVal.
pVal	Returns the current value of OcspTrustedRespSignerCertPath property.

OverwriteFile

Read / Write	Write only.
IDL File Declaration	[propput, id(20), helpstring("property OverwriteFile")] HRESULT OverwriteFile([in] BOOL newVal);
Java Interface	public void setOverwriteFile(int newVal) sigObj.setOverwriteFile (1); // Enable overwrite for saveXMLSignature() method
C Interface	ISignature_put_OverwriteFile (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->put_OverwriteFile (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.OverwriteFile = 1
Supported Platforms	All

When this property it set to 1, it allows SaveXMLSignature method to overwrite destination file if it already exists.

Parameters:

newVal Sets the OverwriteFile property to newVal.

PhysicalSignatureB64Str

Read / Write	Write only.
IDL File Declaration	[propput, id(148), helpstring("property PhysicalSignatureB64Str")] HRESULT PhysicalSignatureB64Str([in] BSTR newVal);
Java Interface	public void setPhysicalSignatureB64Str(String newVal); sigObj.setPhysicalSignatureB64Str (newVal); // Set signature image
C Interface	ISignature_put_PhysicalSignatureB64Str (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_PhysicalSignatureB64Str (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.PhysicalSignatureB64Str = newVal
Supported Platforms	Windows Only

Must be set to a valid b64 encoded signature image data if PhysicalSignatureUsage is set to 3.

Parameters:

newVal Sets the PhysicalSignatureFile to pVal.

PhysicalSignatureFile

Read / Write	Both
IDL File Declaration	[propget, id(13), helpstring("property PhysicalSignatureFile")] HRESULT PhysicalSignatureFile([out, retval] BSTR *pVal); [propput, id(13), helpstring("property PhysicalSignatureFile")] HRESULT PhysicalSignatureFile([in] BSTR newVal);

Java Interface	public String getPhysicalSignatureFile();
	public void setPhysicalSignatureFile(String lastParam);
	String curPhysicalSignatureFileVal = sigObj.getPhysicalSignatureFile();
	sigObj.setPhysicalSignatureFile("C:\\temp\\mysig.gif"); // Set signature image

C Interface	ISignature_get_PhysicalSignatureFile (ISignature *pSig, BSTR *pVal); ISignature_put_PhysicalSignatureFile (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_PhysicalSignatureFile (BSTR *pVal); pSig->put_PhysicalSignatureFile (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.PhysicalSignatureFile = "C:\\temp\\mysig.gif" pVal = sigObj.PhysicalSignatureFile
Supported Platforms	Windows Only

Must be set to a valid signature image file path if PhysicalSignatureUsage is set to 1.

Parameters:

newVal	Sets the PhysicalSignatureFile to pVal.
pVal	Returns the current value of PhysicalSignatureFile

PhysicalSignatureUsage

Read / Write	Both
IDL File Declaration	[propget, id(12), helpstring("property PhysicalSignatureUsage")] HRESULT PhysicalSignatureUsage([out, retval] long *pVal); [propput, id(12), helpstring("property PhysicalSignatureUsage")] HRESULT PhysicalSignatureUsage([in] long newVal);

Java Interface	public int getPhysicalSignatureUsage();
	public void setPhysicalSignatureUsage(int lastParam);
	int curPhysicalSignatureUsageVal = sigObj.getPhysicalSignatureUsage();
	sigObj.setPhysicalSignatureUsage(2); // Set physical signature to live mode

C Interface	ISignature_get_PhysicalSignatureUsage (ISignature *pSig, long *pVal); ISignature_put_PhysicalSignatureUsage (ISignature *pSig, long newVal);
C++ Interface	pSig->get_PhysicalSignatureUsage (long *pVal); pSig->put_PhysicalSignatureUsage (long newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.PhysicalSignatureUsage = 2 pVal = sigObj.PhysicalSignatureUsage
Supported Platforms	Windows Only

Sets the operational mode for the physical signature usage for enveloped signature creation. Support for Wintab Compatible external signature device is provided by Securepad.dll ActiveX component. Wintab32.dll provided by the device manufacturer must be present before installing SecureXML.

Parameters:

newVal Sets the PhysicalSignatureUsage flag to pVal.
The following are the allowed values and their meaning:

Value	Interpretation
0	Don't use physical signature image (default)
1	Use a signature image stored in a file pointed to by PhysicalSigantureFile property
2	Capture live signature. Use a signature pad if present, otherwise use mouse to capture signature

pVal Returns the current settings of PhysicalSignatureUsage

Properties

Read / Write	Both
IDL File Declaration	[propget, id(5), helpstring("property Properties")] HRESULT Properties([in] long sigIndex, [in] long propIndex, [out, retval] BSTR *pVal); [propput, id(5), helpstring("property Properties")] HRESULT Properties([in] long sigIndex, [in] long propIndex, [in] BSTR newVal);

Java Interface	public String getProperties(int sigIndex, int propIndex);
	public void setProperties(int sigIndex, int propIndex, String lastParam);
	String curPropertiesVal = sigObj.getProperties(int sigIndex, int propIndex);
	sigObj.setProperties(0, propIndex ,"Name=John Doe"); // Set signature property

C Interface	ISignature_get_Properties (ISignature *pSig, long sigIndex, long propIndex, BSTR *pVal); ISignature_put_Properties (ISignature *pSig, long sigIndex, long propIndex, BSTR newVal);
C++ Interface	pSig->get_Properties (long sigIndex, long propIndex, BSTR *pVal); pSig->put_Properties (long sigIndex, long propIndex, BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.Properties(0, propIndex) = "Name=John Doe" pVal = sigObj.Properties(sigIndex, propIndex)
Supported Platforms	All

During a write operation, it sets the property for the current signature object. These properties are attached to the <Object> element of the current signature. This is where application specific data can be added to a signature. The demo application uses Properties to add signature date/time to the signature. The format for all properties must be

"Parameter = Value"

A total of up to 32 properties can be added to each signature.

During a read operation it fetches the property string(s) indexed by propIndex for the signature indexed by sigIndex.

Parameters:

sigIndex	During a write operation the first parameter, sigIndex, is ignored. During a read operation this is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount -1.
propIndex	It can vary from 0 to 31. During a write operation the PropertyCount is set to the last index used. So if you set Property #1 first and then #0, the signature will have only one property (the one 'corresponding to #0). During a read operation it refers to the sequence number of the properties contained in the signature referred to by sigIndex. During a read operation, it can vary from 0 to PropertyCount – 1. PropertyCount is obtained by calling GetPropertyCount(sigIndex) method.
pVal	It is string of type "Parameter = Value"

ProxyHost

Read / Write	Write only.
IDL File Declaration	[propput, id(171), helpstring("property ProxyHost")] HRESULT ProxyHost([in] BSTR newVal);
Java Interface	public void setProxyHost(String lastParam); sigObj.setProxyHost (“192.168.0.3”); // Set proxy host
C Interface	ISignature_put_ProxyHost (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_ProxyHost (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ProxyHost = “192.168.0.3”
Supported Platforms	All

This property relevant only for OCSP based certificate validation. If ProxyHost is set, then it is used for reaching the OCSP server. In addition to ProxyHost property, ProxyPort, ProxyUserName and ProxyPassword properties are also used for this operation. The ProxyHost could either be an IP address or an URL.

ProxyPassword

Read / Write	Write only.
IDL File Declaration	[propput, id(174), helpstring("property ProxyPassword")] HRESULT ProxyPassword([in] BSTR newVal);
Java Interface	public void setProxyPassword(String lastParam) sigObj.setProxyPassword (“password”); // Set proxy host password
C Interface	ISignature_put_ProxyPassword (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_ProxyPassword (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ProxyPassword = “password”
Supported Platforms	All

This property relevant only for OCSP based certificate validation. If ProxyHost is set, then it is used for reaching the OCSP server. In addition to ProxyHost property, ProxyPort, ProxyUserName and ProxyPassword properties are also used for this operation. The ProxyHost could either be an IP address or an URL. Proxy password must be set to the right password for the user provided in the ProxyPassword property.

ProxyPort

Read / Write	Write only.
IDL File Declaration	[propput, id(172), helpstring("property ProxyPort")] HRESULT ProxyPort([in] USHORT newVal);
Java Interface	public void setProxyPort(int lastParam); sigObj.setProxyPort (8080); // Set proxy port
C Interface	ISignature_put_ProxyPort (ISignature *pSig, USHORT newVal);
C++ Interface	pSig->put_ProxyPort (USHORT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ProxyPort = 8080
Supported Platforms	All

This property relevant only for OCSP based certificate validation. If ProxyHost is set, then it is used for reaching the OCSP server. In addition to ProxyHost property, ProxyPort, ProxyUserName and ProxyPassword properties are also used for this operation. The ProxyHost could either be an IP address or an URL. The default proxy port is 80. The ProxyPort should be set to the port on which the proxy server is listening for incoming connections.

ProxyUserName

Read / Write	Write only.
IDL File Declaration	[propput, id(173), helpstring("property ProxyUserName")] HRESULT ProxyUserName([in] BSTR newVal);
Java Interface	public void setProxyUserName(String lastParam); sigObj.setProxyUserName ("guest"); // Set proxy user name
C Interface	ISignature_put_ProxyUserName(ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_ProxyUserName (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ProxyUserName = "guest"
Supported Platforms	All

This property relevant only for OCSP based certificate validation. If ProxyHost is set, then it is used for reaching the OCSP server. In addition to ProxyHost property, ProxyPort, ProxyUserName and ProxyPassword properties are also used for this operation. The ProxyHost could either be an IP address or an URL. The default proxy port is 80. The ProxyPort should be set to the port on which the proxy server is listening for incoming connections. If the proxy server requires authentication, ProxyUserName should be set to the user making the proxy request. Corresponding ProxyPassword must also be set in order for the OCSP request to go past the proxy server.

RecipientCertificateFiles

Read / Write	Write Only
IDL File Declaration	[propput, id(98), helpstring("property RecipientCertificateFiles")] HRESULT RecipientCertificateFiles([in] VARIANT newVal);
Java Interface	public void setRecipientCertificateFiles (String [] lastParam); String [] recipientCertificateFileList = new String[3]; recipientCertificateFileList[0] = "c:\myRecipientCertificateFiles\john.cer"; recipientCertificateFileList[1] = "http://www.infomosaic.com/jane.cer"; recipientCertificateFileList[2] = "ldap://myldapserver.ldap.com/doe.crt"; sigObj.setRecipientCertificateFiles (recipientCertificateFileList);
C Interface	ISignature_put_RecipientCertificateFiles (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_RecipientCertificateFiles (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim recipientCertificateFileList(3) recipientCertificateFileList[0] = "c:\myRecipientCertificateFiles\john.cer"; recipientCertificateFileList[1] = "http://www.infomosaic.com/jane.cer"; recipientCertificateFileList[2] = "ldap://myldapserver.ldap.com/doe.crt"; sigObj.RecipientCertificateFiles = recipientCertificateFileList Or string [] recipientCertificateFileList = new String[3]; recipientCertificateFileList[0] = "c:\myRecipientCertificateFiles\john.cer"; recipientCertificateFileList[1] = "http://www.infomosaic.com/jane.cer"; recipientCertificateFileList[2] = "ldap://myldapserver.ldap.com/doe.crt"; sigObj.RecipientCertificateFiles = recipientCertificateFileList
Supported Platforms	Windows Only

This property is used by the EncryptStr and EncryptFile methods. The encrypted data uses the public key contained in the certificates pointed to by the list provided. Please note that if the certificate files contain base64 encoded certificate, they must **not** contain any leading or trailing markers such as "----------BEGIN CERTIFICATE----------" or "----------END CERTIFICATE----------". The behavior would be unpredictable if they do.

If neither RecipientCertificateFiles nor RecipientCertificates is set, a certificate selection dialog box is presented and it allows for selecting one recipient certificate for the encrypted content.

Parameters:

newVal The given recipient certificate files.

RecipientCertificates

Read / Write	Write Only
IDL File Declaration	[propput, id(92), helpstring("property RecipientCertificates")] HRESULT RecipientCertificates([in] VARIANT newVal);

Java Interface	public void setRecipientCertificates (String [] lastParam);
	String [] recipientCertificateList = new String[3];
	recipientCertificateList[0] = cert1; //cert1, cert2 and cert3 contain base64 encoded certs
	recipientCertificateList[1] = cert2;
	recipientCertificateList[2] = cert2;
	sigObj.setRecipientCertificates (recipientCertificateList);

C Interface	ISignature_put_RecipientCertificates (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_RecipientCertificates (VARIANT newVal);

VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim recipientCertificateList(3)
	recipientCertificateList[0] = cert1; 'cert1, cert2 and cert3 contain base64 encoded certs
	recipientCertificateList[1] = cert2;
	recipientCertificateList[2] = cert2;
	sigObj.RecipientCertificates = recipientCertificateList
	Or
	string [] recipientCertificateList = new String[3];
	recipientCertificateList[0] = cert1; //cert1, cert2 and cert3 contain base64 encoded certs
	recipientCertificateList[1] = cert2;
	recipientCertificateList[2] = cert2;
	sigObj.RecipientCertificates = recipientCertificateList

Supported Platforms	Windows Only

This property is used by the EncryptStr and EncryptFile methods. The encrypted data uses the public key contained in the base64 encoded certificates contained in the buffer pointed to by the list provided. Please note that the certificates must **not** contain any leading or trailing markers such as "----------BEGIN CERTIFICATE----------" or "----------END CERTIFICATE----------". The behavior would be unpredictable if they do.

If neither RecipientCertificates nor RecipientCertificates is set, a certificate selection dialog box is presented and it allows for selecting one recipient certificate for the encrypted content.

Parameters:

newVal The given recipient certificates.

RecipientCertificateStore

Read / Write	Both
IDL File Declaration	[propget, id(2), helpstring("property RecipientCertificateStore")] HRESULT RecipientCertificateStore([out, retval] BSTR *pVal); [propput, id(2), helpstring("property RecipientCertificateStore")] HRESULT RecipientCertificateStore([in] BSTR newVal);

Java Interface	public String getRecipientCertificateStore();
	public void setRecipientCertificateStore(String lastParam);
	String curRecipientCertificateStoreVal = sigObj.getRecipientCertificateStore ();
	sigObj.setRecipientCertificateStore ("MY"); // Set the certificate store from which to show
	// the certificate selection window.

C Interface	ISignature_get_RecipientCertificateStore (ISignature *pSig, BSTR *pVal); ISignature_put_RecipientCertificateStore (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_RecipientCertificateStore (BSTR *pVal); pSig->put_RecipientCertificateStore (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.RecipientCertificateStore = "MY" pVal = sigObj.RecipientCertificateStore
Supported Platforms	Windows Only

By default the recipient certificate store is the current user's "addressbook" store. You can change that by setting this property. If neither RecipientCertificates nor RecipientCertificateFiles is set, a dialog is displayed for selecting the recipient certificate during a call to the EncryptStr and EncryptFile methods. The dialog displays the certificates contained in the RecipientCertificateStore.

Parameters:

newVal	Sets the RecipientCertificateStore to pVal.
pVal	Returns the current value of RecipientCertificateStore

SecureXMLPath

Read / Write	Read Only
IDL File Declaration	[propget, id(127), helpstring("property SecureXMLPath")] HRESULT SecureXMLPath([out, retval] BSTR* pVal);
Java Interface	public String getSecureXMLPath (); String curSecureXMLPathVal = sigObj.getSecureXMLPath ();
C Interface	ISignature_get_SecureXMLPath (ISignature *pSig, BSTR *pVal);
C++ Interface	pSig->get_SecureXMLPath (BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SecureXMLPath
Supported Platforms	All

Returns the absolute path of the XMLSign.dll file.

Parameters:

pVal Returns the absolute path of the XMLSign.dll file.

SigCertStatus

Read / Write	Read Only.
IDL File Declaration	[propget, id(65), helpstring("property SigCertStatus")] HRESULT SigCertStatus([in] long sigIndex, [in] BSTR atTime, [in] long timeFormat, [out, retval] long *pVal);
Java Interface	public int getSigCertStatus(int sigIndex, String atTime, int timeFormat); int curSigCertStatusVal = sigObj.getSigCertStatus(sigIndex, atTime, timeFormat);
C Interface	ISignature_get_SigCertStatus (ISignature *pSig, long sigIndex, BSTR atTime, long timeFormat, long *pVal);
C++ Interface	pSig->get_SigCertStatus (long sigIndex, BSTR atTime, long timeFormat, long *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SigCertStatus(sigIndex, atTime, timeFormat)
Supported Platforms	All

This property can be invoked only after a call to one of the signature verification methods.

Parameters:

sigIndex	This is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount - 1.
atTime	This contains a valid time in the format indicated by timeFormat parameter. This may be NULL.
timeFormat	The valid values are 0, 1 and 2. They correspond to TIME_RFC, TIME_VB_NOW and TIME_JS_UTC time formats. For details please refer to the documentation for ***VerifyPFXCertCRL*** method later in this guide.
pVal	On return *pVal is set to CERT_TRUST_ERROR if the certificate was revoked at signature time or atTime. It is set to 0 (zero) otherwise.
Supported Platforms	All

GetLastError can return the following error codes after a Read operation:

SIG_INDEX_ERROR	sigIndex provided is >= SignatureCount or < 0.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

SignatureID

Read / Write	Both
IDL File Declaration	[propget, id(4), helpstring("property SignatureID")] HRESULT SignatureID([in]long index, [out, retval] BSTR *pVal); [propput, id(4), helpstring("property SignatureID")] HRESULT SignatureID([in]long index, [in] BSTR newVal);

Java Interface	public String getSignatureID(int index);
	public void setSignatureID(int index, String lastParam);
	String curSignatureIDVal = sigObj.getSignatureID(int index);
	sigObj.setSignatureID(0, "MySignature"); // Set signature Id

C Interface	ISignature_get_SignatureID (ISignature *pSig, long index, BSTR *pVal); ISignature_put_SignatureID (ISignature *pSig, long index, BSTR newVal);
C++ Interface	pSig->get_SignatureID (long index, BSTR *pVal); pSig->put_SignatureID (long index, BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SignatureID(0) = "MySignature" pVal = sigObj.SignatureID(index)
Supported Platforms	All

Parameters:

index	During a read operation, this is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount - 1. During the write operation this parameter is ignored.
pVal	On return *pVal is set to the corresponding signature ID.
newVal	Sets the signature ID to this value for subsequent signature creation operation.

GetLastError can return the following error codes after a Read operation:

SIG_INDEX_ERROR	index provided is >= SignatureCount or < 0.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

SignatureImageId

Read / Write	Read/Write
IDL File Declaration	[propget, id(152), helpstring("property SignatureImageId")] HRESULT SignatureImageId([out, retval] BSTR* pVal); [propput, id(152), helpstring("property SignatureImageId")] HRESULT SignatureImageId([in] BSTR newVal);
Java Interface	public String getSignatureImageId(); public void setSignatureImageId(String lastParam);
C Interface	ISignature_get_SignatureImageId (ISignature *pSig, BSTR *pVal); ISignature_put_SignatureImageId (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_SignatureImageId (BSTR *pVal); pSig->put_SignatureImageId (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SignatureImageId sigObj.SignatureImageId = newVal
Supported Platforms	Windows Only

This property is part of the split signing mechanism of SecureXML. After capturing a live/file signature image on the client side, get the SignatureImageId and pass it to the server side along with the Base64 encoded signature image. The server side code would need to set PhysicalSignatureUsage = 3, assign SignatureImageId and the base64 encoded signature image data to the SignatureImageId and PhysicalSignatureB64Str respectively before calling GetSignedInfoDigest or GetSignedInfoDigestFromByteArray.

Parameters:

newVal	The value to set.
pVal	The signature image id of the recently captured signature image.

SignatureIndexToVerify

Read / Write	Read/Write
IDL File Declaration	[propget, id(161), helpstring("property SignatureIndexToVerify")] HRESULT SignatureIndexToVerify([out, retval] LONG* pVal); [propput, id(161), helpstring("property SignatureIndexToVerify")] HRESULT SignatureIndexToVerify([in] LONG newVal);

Java Interface	public int getSignatureIndexToVerify ();
	public void setSignatureIndexToVerify (int lastParam);
	int curSignatureIndexToVerify = sigObj.getSignatureIndexToVerify ();
	sigObj.setSignatureIndexToVerify (2); // Make SecureXML verify the 3rd signature
	// in the input signed XML during a call to signature
	// verification methods

C Interface	ISignature_get_SignatureIndexToVerify (ISignature *pSig, BOOL *pVal); ISignature_put_SignatureIndexToVerify (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_SignatureIndexToVerify (BOOL *pVal); pSig->put_SignatureIndexToVerify (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SignatureIndexToVerify = 2 pVal = sigObj.SignatureIndexToVerify
Supported Platforms	All

By default SecureXML attempts to verify each and every signature element found in the input document during any of the signature verification methods. If SignatureIndexToVerify is set to either 0 or a positive integer value < SignatureCount, only the signature element referred by SignatureIndexToVerify value is verified during a call to any of the signature verification methods. If IgnoreIncompleteSignature property is set, SignatureIndexToVerify refers to the index of complete signature elements. The default value for SignatureIndexToVerify is -1, which forces all signatures to be verified.

Parameters:

pVal	Returns the current setting for the SignatureIndexToVerify property.
newVal	Sets the SignatureIndexToVerify property. It can vary from -1 to SignatureCount – 1.

SignatureStatus

Read / Write	Read Only
IDL File Declaration	[propget, id(11), helpstring("property SignatureStatus")] HRESULT SignatureStatus([in] long sigIndex, [out, retval] BOOL *pVal);
Java Interface	public int getSignatureStatus(int sigIndex); int curSignatureStatusVal = sigObj.getSignatureStatus(int sigIndex);
C Interface	ISignature_get_SignatureStatus (ISignature *pSig, long sigIndex, BOOL *pVal);
C++ Interface	pSig->get_SignatureStatus (long sigIndex, BOOL *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SignatureStatus(sigIndex)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default).

Parameters:

sigIndex	This is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount -1.
pVal	*pVal is 1 if signature was verified successfully. It is 0 otherwise.

GetLastError can return the following error codes:

SIG_INDEX_ERROR	sigIndex provided is >= SignatureCount or < 0.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

SignedDocumentCount

Read / Write	Read only
IDL File Declaration	[propget, id(128), helpstring("property SignedDocumentCount")] HRESULT SignedDocumentCount([in] LONG sigIndex, [out, retval] LONG* pVal);
Java Interface	public int getSignedDocumentCount(int sigIndex); int pVal = sigObj.getSignedDocumentCount(sigIndex);
C Interface	ISignature_get_SignedDocumentCount(ISignature *pSig, long sigIndex, long *pVal);
C++ Interface	pSig->get_SignedDocumentCount(long sigIndex, long *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SignedDocumentCount(sigIndex)
Supported Platforms	All

Valid only after a successful call to a signature verification method.

Parameters:

sigIndex	This is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount -1.
pVal	On return *pVal is set to the number of references signed by the signer referred by sigIndex.

GetLastError can return the following error codes:

SIG_INDEX_ERROR	sigIndex provided is >= SignatureCount or < 0.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

SignedDocumentPath

Read / Write	Read only
IDL File Declaration	[propget, id(29), helpstring("property SignedDocumentPath")] HRESULT SignedDocumentPath([in] long sigIndex, [in] long uriIndex, [out, retval] BSTR *pVal);
Java Interface	public String getSignedDocumentPath(int sigIndex, int uriIndex); String signedDocPath = sigObj.getSignedDocumentPath(sigIndex, uriIndex);
C Interface	ISignature_get_SignedDocumentPath(ISignature *pSig, long sigIndex, long uriIndex, BSTR *pVal);
C++ Interface	pSig->get_SignedDocumentPath(long sigIndex, long uriIndex, BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SignedDocumentPath(sigIndex, uriIndex)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default).

Parameters:

sigIndex	This is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount -1.
uriIndex	This is the index number for the signed URI in the signature corresponding to sigIndex. A uriIndex can be from 0 to TotalUriCount().
pVal	On return *pVal is set to the URI where one can access the signed object from.

GetLastError can return the following error codes:

SIG_INDEX_ERROR	sigIndex provided is >= SignatureCount or < 0.
URI_INDEX_ERROR	uriIndex provided is >= maxURI where maxURI is the number of URI referred to by the current signature
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.
DOC_PATH_NOT_FOUND	There was no document path information for the given sigIndex/uriIndex combination

---

SignerCertificate

Read / Write	Both
IDL File Declaration	[propget, id(3), helpstring("property SignerCertificate")] HRESULT SignerCertificate([in] long index, [out, retval] BSTR *pVal); [propput, id(3), helpstring("property SignerCertificate")] HRESULT SignerCertificate([in] long index, [in] BSTR newVal);

Java Interface	public String getSignerCertificate(int index);
	public void setSignerCertificate(int index, String lastParam);
	String curSignerCertificateVal = sigObj.getSignerCertificate(int index);
	sigObj.setSignerCertificate(0, "MySignature"); // Set signature Id

C Interface	ISignature_get_SignerCertificate (ISignature *pSig, long index, BSTR *pVal); ISignature_put_SignerCertificate (ISignature *pSig, long index, BSTR newVal);
C++ Interface	pSig->get_SignerCertificate (long index, BSTR *pVal); pSig->put_SignerCertificate (long index, BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SignerCertificate(0) = stringContainingBase64EncodedX509Certificate pVal = sigObj.SignerCertificate(index)
Supported Platforms	All

Use this property to either extract the base64 encoded certificate from the signature just verified or to set the signer certificate for subsequent signature creation operation. You can invoke SetActiveCertificate, SelectActiveCertificate or SetActivePFXFileCert methods to override this value. Also you can invoke SignerCertificate (put) method with newVal set to NULL to force SecureXML to show you a certificate selection dialog box.

A typical use case for setting this property to a base64 encoded certificate would be when you want to store user certificates on a central location such as an LDAP or database storage and keep the user private keys on smart cards or other hardware tokens. This way the user can be completely mobile as installation of the certificate in the local windows certificate store is not required.

Parameters:

index	During a read operation, this is the signature index of a signature which was verified previously. A signature index can be from 0 to SignatureCount - 1. During the write operation this parameter is ignored.
pVal	On return *pVal is set to the corresponding signature creator's base64 encoded certificate.
newVal	Sets the signer certificate to this base64 encoded value for subsequent signature creation operations. The certificate is an X.509 certificate with only the public key (it is not a pfx or p12 file data).

GetLastError can return the following error codes after a Read operation:

SIG_INDEX_ERROR	index provided is >= SignatureCount or < 0.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

SignerCertificateChain

Read / Write	Write only
IDL File Declaration	[propput, id(151), helpstring("property SignerCertificateChain")] HRESULT SignerCertificateChain([in] BSTR newVal);
Java Interface	public void setSignerCertificateChain(String lastParam); sigObj.setSignerCertificateChain(certChainData); // Set signer certificate chain
C Interface	ISignature_put_SignerCertificateChain (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->put_SignerCertificateChain (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SignerCertificateChain = stringContainingOutputFromGetX509CertificateChain
Supported Platforms	All

This property is part of the split signing mechanism of SecureXML where the signed info digest is calculated on the server side while the signature value calculation is performed on the client side. The client side code calls GetX509CertificateChain and sends the return value to the server side. The server side application uses this value to set the SignerCertificateChain property before calling GetSignedInfoDigest or GetSignedInfoDigestFromByteArray methods. The GetSignedInfoDigest and GetSignedInfoFromByteArray methods will perform certificate validation as per the object settings on the server side on the certificate information provided by SignerCertificateChain property. The setting of the SignerCertificateChain is a required step for a successful call to GetSignedInfoDigest or GetSignedInfoDigestFromByteArray methods since the signature algorithm, which is part of the signed info, depends on the certificate type (RSA/DSA) to be used for signing the signed info digest.

Parameters:

newVal	Sets the signer certificate chain to this value, which should be the output of a previous call to GetX509CertificateChain method.

SignerSubject

Read / Write	Read Only
IDL File Declaration	[propget, id(30), helpstring("property SignerSubject")] HRESULT SignerSubject([in] BSTR sigId, [out, retval] BSTR *pVal);
Java Interface	public String getSignerSubject(String sigId); String SignerSubject = sigObj.getSignerSubject(sigId);
C Interface	ISignature_get_SignerSubject(ISignature *pSig, BSTR sigId, BSTR *pVal);
C++ Interface	pSig->get_SignerSubject(BSTR sigId, BSTR *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SignerSubject(sigId)
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default).

Parameters:

sigId	This is the signature ID of a signature which was verified previously. A signature ID is obtained by calling SignatureID() property getter.
pVal	On return *pVal is set to the subject name contained in the certificate used for signing,

GetLastError can return the following error codes:

SIG_NOT_FOUND	A signature with matching sigId was not found in the signature just verified.
NO_SIGNATURE_DATA	Either no signature verification was performed or there was no XML signature in the input XML which was just verified.

SignatureCount

Read / Write	Read only
IDL File Declaration	[propget, id(39), helpstring("property SignatureCount")] HRESULT SignatureCount([out, retval] long *pVal);
Java Interface	public int getSignatureCount(); int certCount = sigObj.getSignatureCount();
C Interface	ISignature_get_SignatureCount(ISignature *pSig, long *pVal);
C++ Interface	pSig->get_SignatureCount(long *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.SignatureCount
Supported Platforms	All

Valid only after a successful call to Verify () or VerifyXMLStr() methods with DetailedVerificationFlag set to 1 (default).

Parameters:

pVal	On return *pVal is set to the total number of signatures found in the XML Signature verified previously.

TimeStampURL

Read / Write	Read/Write
IDL File Declaration	[propget, id(61), helpstring("property TimeStampURL")] HRESULT TimeStampURL([out, retval] BSTR *pVal); [propput, id(61), helpstring("property TimeStampURL")] HRESULT TimeStampURL([in] BSTR newVal);

Java Interface	public String getTimeStampURL();
	public void setTimeStampURL(String lastParam);
	String curTimeStampURLVal = sigObj.getTimeStampURL();
	sigObj.setTimeStampURL("http://www.mytimestampserver.com");

C Interface	ISignature_get_TimeStampURL (ISignature *pSig, BSTR *pVal); ISignature_put_TimeStampURL (ISignature *pSig, BSTR newVal);
C++ Interface	pSig->get_TimeStampURL (BSTR *pVal); pSig->put_TimeStampURL (BSTR newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.TimeStampURL = "http://www.mytimestampserver.com" pVal = sigObj.TimeStampURL
Supported Platforms	All

It sets the URL used for obtaining the current time. The default time server URL is set to http://time-b.timefreq.bldrdoc.gov:13. Time stamping must be enabled by setting TimeStamping to 1. Time stamping is disabled by default. Set TimeStampURL to "SystemTime" in order to use the local machines system time for time stamping. The time stamp is always added in UTC.

Parameters:

pVal	On return *pVal is set to the current time stamp URL.
newVal	Assigns the time stamp URL to the input value for all subsequent signature creations.

TimeStampCritical

Read / Write	Read/Write
IDL File Declaration	[propget, id(177), helpstring("property TimeStampCritical")] HRESULT TimeStampCritical([out, retval] LONG* pVal); [propput, id(177), helpstring("property TimeStampCritical")] HRESULT TimeStampCritical([in] LONG newVal);

Java Interface	public int getTimeStampCritical();
	public void setTimeStampCritical(int lastParam);
	int curTimeStampCriticalVal = sigObj.getTimeStampCritical();
	sigObj.setTimeStampCritical(1); // Enable Time Stamp Critical

C Interface	ISignature_get_TimeStampCritical (ISignature *pSig, BOOL *pVal); ISignature_put_TimeStampCritical(ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_TimeStampCritical (BOOL *pVal); pSig->put_TimeStampCritical (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.TimeStampCritical = 1 pVal = sigObj.TimeStampCritical
Supported Platforms	All

Enables or disables time stamp as a critical requirement for signature creation. By default TimeStampCritical is disabled (=0). If TimeStampCritical is set to 1, and TimeStamping is enabled (i.e. set to 1 or 2), a failure to obtain time stamp leads to signature creation failure.

Parameters:

pVal	Returns the current setting for the time stamp critical flag.
newVal	Sets the time stamp critical flag. A zero value disables time stamp critical and a one value enables it.

TimeStampFormat

Read / Write	Read/Write
IDL File Declaration	[propget, id(176), helpstring("property TimeStampFormat")] HRESULT TimeStampFormat([out, retval] LONG* pVal); [propput, id(176), helpstring("property TimeStampFormat")] HRESULT TimeStampFormat([in] LONG newVal);

Java Interface	public int getTimeStampFormat();
	public void setTimeStampFormat(int lastParam);
	int curTimeStampFormatVal = sigObj.getTimeStampFormat();
	sigObj.setTimeStampFormat(1); // Set Time Stamp Format value

C Interface	ISignature_get_TimeStampFormat (ISignature *pSig, BOOL *pVal); ISignature_put_TimeStampFormat(ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_TimeStampFormat (BOOL *pVal); pSig->put_TimeStampFormat (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.TimeStampFormat = 1 pVal = sigObj.TimeStampFormat
Supported Platforms	All

Sets or gets time stamp format value. By default TimeStampFormat is 0 (= RFC-867 Day Time Protocol Format).

Parameters:

pVal Returns the current setting for the time stamp critical flag.

newVal Sets the time stamp critical flag. A zero value disables time stamp critical and a one value enables it.

newVal = 0 for Day Time Protocol (RFC 867) Format
newVal = 1 for VB Now() format: 07/24/2002 3:44:13 PM
newVal = 2 for As returned by JavaScript toUTCString method on Date object: Wed, 24 Jul 2002 22:44:12 UTC

TimeStamping

Read / Write	Read/Write
IDL File Declaration	[propget, id(62), helpstring("property TimeStamping")] HRESULT TimeStamping([out, retval] BOOL *pVal); [propput, id(62), helpstring("property TimeStamping")] HRESULT TimeStamping([in] BOOL newVal);

Java Interface	public int getTimeStamping();
	public void setTimeStamping(int lastParam);
	int curTimeStampingVal = sigObj.getTimeStamping();
	sigObj.setTimeStamping(1); // Enable Time Stamping

C Interface	ISignature_get_TimeStamping (ISignature *pSig, BOOL *pVal); ISignature_put_TimeStamping (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_TimeStamping (BOOL *pVal); pSig->put_TimeStamping (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.TimeStamping = 1 pVal = sigObj.TimeStamping
Supported Platforms	All

Enables or disables time stamping feature. By default TimeStamping is disabled.

Parameters:

pVal	Returns the current setting for the time stamping flag.
newVal	Sets the time stamping flag. A zero value disables time stamping and non-zero value enables it. newVal = 1 for Day Time Protocol (RFC 867) newVal = 2 for SNTP (RFC-1305), Must set TimeStampURL object property. Uses port 123. Must configure firewall to allow outbound UDP traffic on port 123.

TrustedRoots

Read / Write	Write Only
IDL File Declaration	[propput, id(97), helpstring("property TrustedRoots")] HRESULT TrustedRoots([in] VARIANT newVal);
Java Interface	public void setTrustedRoots (String [] lastParam); String [] trustedRootList = new String[3]; trustedRootList[0] = "c:\myTrustedRoots\trustedRoot1.cer"; trustedRootList[1] = "http://www.infomosaic.com/TrustedRootfile.crt "; trustedRootList[2] = "ldap://myldapserver.ldap.com/trustedRoot.cer"; sigObj.setTrustedRoots (trustedRootList);
C Interface	ISignature_put_TrustedRoots (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_TrustedRoots (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim trustedRootList(3) trustedRootList[0] = "c:\myTrustedRoots\trustedRoot1.cer"; trustedRootList[1] = "http://www.infomosaic.com/TrustedRootfile.crt "; trustedRootList[2] = "ldap://myldapserver.ldap.com/trustedRoot.cer"; sigObj.TrustedRoots = trustedRootList Or string [] trustedRootList = new String[3]; trustedRootList[0] = "c:\myTrustedRoots\trustedRoot1.cer"; trustedRootList[1] = "http://www.infomosaic.com/TrustedRootfile.crt "; trustedRootList[2] = "ldap://myldapserver.ldap.com/trustedRoot.cer"; sigObj.TrustedRoots = trustedRootList
Supported Platforms	All

If the CertificateTrustExplicit is enabled, the certificate validation succeeds only if the root certificate issuer is one of the entries provided by the TrustedRoots property.

Parameters:

newVal The given TrustedRoots locations.

UseCam

Read / Write	Read/Write
IDL File Declaration	[propget, id(108), helpstring("property UseCam")] HRESULT UseCam([out, retval] BOOL* pVal); [propput, id(108), helpstring("property UseCam")] HRESULT UseCam([in] BOOL newVal);

Java Interface	public int getUseCam();
	public void setUseCam(int lastParam);
	int curUseCamVal = sigObj.getUseCam();
	sigObj.setUseCam(1); // Enable CAM based certificate validation

C Interface	ISignature_get_UseCam (ISignature *pSig, BOOL *pVal); ISignature_put_UseCam (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_UseCam (BOOL *pVal); pSig->put_UseCam (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.UseCam = 1 pVal = sigObj.UseCam
Supported Platforms	All

Enables or disables CAM Server usage for certificate validation. Set UseCam to 1 to enable CAM Server usage for certificate validation.

In addition to setting UseCam, the application must also set the appropriate values for the CamServerHost and CamServerPort properties.

Parameters:

pVal	Returns the current setting for the UseCam flag.
newVal	Sets the UseCam flag. A zero value disables CAM Server based certificate validation and 1 value enables it.

UseCRLCache

Read / Write	Read/Write
IDL File Declaration	[propget, id(122), helpstring("property UseCRLCache")] HRESULT UseCRLCache([out, retval] BOOL* pVal); [propput, id(122), helpstring("property UseCRLCache")] HRESULT UseCRLCache([in] BOOL newVal);

Java Interface	public int getUseCRLCache() ;
	public void setUseCRLCache(int lastParam);
	int curUseCRLCacheVal = sigObj.getUseCRLCache ();
	sigObj.setUseCRLCache (1); // Enable CRL Caching

C Interface	ISignature_get_UseCRLCache (ISignature *pSig, BOOL *pVal); ISignature_put_UseCRLCache (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_UseCRLCache (BOOL *pVal); pSig->put_UseCRLCache (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.UseCRLCache = 1 pVal = sigObj.UseCRLCache
Supported Platforms	Windows Only

This property enables or disables CRL Caching. The default value is 0 (zero) or disabled. If it is set to a non-zero value, CRL checking gets enabled for all calls to signature creation and verification methods, provided CrlChecking is also enabled. The database specified by the current value of CRLCacheDbConnectionString is used for CRL storage.

It is recommended that GetLastError or GetError be called immediately after setting UseCRLCache to 1. If there was any problem accessing the CRL Database, appropriate error values are returned.

Once enabled, SecureXML manages the CRL Cache database based on the NextUpdate time contained in each of the fetched CRLs. If the current time is greater than the NextUpdate time, the cached CRL is deleted from the database and a new CRL is fetched from the CRL distribution point.

Parameters:

newVal	Sets the UseCRLCache property to newVal
pVal	On return *pVal is set to the current setting for this property.

UsedCRLList

Read / Write	Read Only
IDL File Declaration	[propget, id(146), helpstring("property UsedCRLList")] HRESULT UsedCRLList([out, retval] VARIANT* pVal);
Java Interface	public String [] getUsedCRLList (); String [] pVal = sigObj.getUsedCRLList ();
C Interface	ISignature_get_ UsedCRLList (ISignature *pSig, VARIANT *usedCRLListVariant);
C++ Interface	pSig->get_UsedCRLList (VARIANT * usedCRLListVariant);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Set usedCRLListVariant = sigObj.UsedCRLList Or string [] usedCRLStringArray = (string [])sigObj. UsedCRLList;
Supported Platforms	All

This property returns the CRL data used during the last certificate validation performed either during signature creation or verification.

Parameters:

pVal Returns the CRL data list as a VARIANT.

UseHMAC

Read / Write	Read/Write
IDL File Declaration	[propget, id(82), helpstring("property UseHMAC")] HRESULT UseHMAC([out, retval] BOOL *pVal); [propput, id(82), helpstring("property UseHMAC")] HRESULT UseHMAC([in] BOOL newVal);

Java Interface	public int getUseHMAC();
	public void setUseHMAC(int lastParam);
	int curUseHMACVal = sigObj.getUseHMAC();
	sigObj.setUseHMAC(1); // Enable HMAC based signature creation

C Interface	ISignature_get_UseHMAC (ISignature *pSig, BOOL *pVal); ISignature_put_UseHMAC (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_UseHMAC (BOOL *pVal); pSig->put_UseHMAC (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.UseHMAC = 1 pVal = sigObj.UseHMAC
Supported Platforms	All

Enables or disables HMAC signature creation. Set it to 1 to enable HMAC signature creation. It works only for enveloped signatures. That is it has no impact when invoking the **Sign** method with EnvelopingFlag set to either 0 (default) or 1.

Parameters:

pVal	Returns the current setting for the UseHMAC flag.
newVal	Sets the UseHMAC flag. A zero value disables HMAC signature creation and 1 value enables it.

UseOcsp

Read / Write	Read/Write
IDL File Declaration	[propget, id(162), helpstring("property UseOcsp")] HRESULT UseOcsp([out, retval] BOOL* pVal); [propput, id(162), helpstring("property UseOcsp")] HRESULT UseOcsp([in] BOOL newVal);

Java Interface	public int getUseOcsp();
	public void setUseOcsp(int lastParam);
	int curUseOcspVal = sigObj.getUseOcsp();
	sigObj.setUseOcsp(1); // Enable CAM based certificate validation

C Interface	ISignature_get_UseOcsp (ISignature *pSig, BOOL *pVal); ISignature_put_UseOcsp (ISignature *pSig, BOOL newVal);
C++ Interface	pSig->get_UseOcsp (BOOL *pVal); pSig->put_UseOcsp (BOOL newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.UseOcsp = 1 pVal = sigObj.UseOcsp
Supported Platforms	All

Enables or disables OCSP for certificate validation. Set UseCam to 1 to enable OCSP usage for certificate validation.

In addition to setting UseOcsp, the application must also set the appropriate values for the other OCSP related properties.

Parameters:

pVal	Returns the current setting for the UseOcsp flag.
newVal	Sets the UseOcsp flag. A zero value disables OCSP based certificate validation and 1 value enables it.

UserConstrainedPolicy

Read / Write	Read Only
IDL File Declaration	[propget, id(73), helpstring("property UserConstrainedPolicy")] HRESULT UserConstrainedPolicy([out, retval] VARIANT *pVal);
Java Interface	public String [] getUserConstrainedPolicy(); String [] userConstPolVar = sigObj.getUserConstrainedPolicy();
C Interface	ISignature_get_UserConstrainedPolicy(ISignature *pSig, VARIANT *userConstPolSet);
C++ Interface	pSig->get_UserConstrainedPolicy (VARIANT *userConstPolSet);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Set userConstPolSet = sigObj.UserConstrainedPolicy Or string [] userConstPolSet = (string [])sigObj.UserConstrainedPolicy;
Supported Platforms	All

This property returns the user constrained policy set after a certificate chain policy verification.

Parameters:

pVal Returns the user constrained policy set after a certificate chain policy verification as a VARIANT.

GetLastError can return the following error codes:

None.

XpathNamespace

Read / Write	Write Only
IDL File Declaration	[propput, id(105), helpstring("property XpathNamespace")] HRESULT XpathNamespace([in] VARIANT newVal);
Java Interface	public void setXpathNamespace (String [] lastParam); String [] xpathNamespaceList = new String[2]; xpathNamespaceList [0] = "nspre0=http://www.infomosaic.net/VerifyResponse.htm"; xpathNamespaceList [1] = "nspre1=http://www.infomosaic.com"; sigObj.setXpathNamespace (xpathNamespaceList);
C Interface	ISignature_put_XpathNamespace (ISignature *pSig, VARIANT newVal);
C++ Interface	pSig->put_XpathNamespace (VARIANT newVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	Dim xpathNamespaceList (2) xpathNamespaceList [0] = "nspre0=http://www.infomosaic.net/VerifyResponse.htm"; xpathNamespaceList [1] = "nspre1=http://www.infomosaic.com"; sigObj.XpathNamespace = xpathNamespaceList Or string [] xpathNamespaceList = new String[2]; xpathNamespaceList [0] = "nspre0=http://www.infomosaic.net/VerifyResponse.htm"; xpathNamespaceList [1] = "nspre1=http://www.infomosaic.com"; sigObj.XpathNamespace = xpathNamespaceList
Supported Platforms	All

This property is relevant to SignXMLXpathStr method only and provides for the namespaces to be used for evaluating the given XPATH expression. All namespaces must have a namespace prefix even if the input XML does not have one. If the input XML does not have a namespace prefix then you can just makeup an arbitrary prefix and use it both here and in the XPATH expression that you specify in the SignXMLXpathStr method. If the input XML has a namespace prefix, you must use the same prefix here and **MUST NOT** use this prefix in your XPATH expression specified in the SignXMLXpathStr method.

Example when the input XML does not have a namespace prefix:

```
sigObj.XpathNamespace = "pre1=http://www.infomosaic.com"
res1 = sigObj.SignXMLXpathStr(xmlStr, "/pre1:MyElementName/*", "MySignature")
```

In the above example the prefix pre1 is arbitrary. If there was a namespace prefix in the input XML, you would use that prefix in the XpathNamespace property and not use it in the Xpath expression as shown below:

```
sigObj.XpathNamespace = "pre1=http://www.infomosaic.com"
res1 = sigObj.SignXMLXpathStr(xmlStr, "/MyElementName/*", "MySignature")
```

At present multiple namespaces are not supported hence you may only specify one and only one namespace when setting the XpathNamespace property.

Future releases will support multiple namespaces and hence the interface has a VARIANT input and the examples above show how it will work when fully implemented.

Parameters:

newVal	The given namespaces to be used for evaluating XPATH expression during a call to SignXMLXpathStr method.

Object Methods

ApplySignatureValue

IDL File Declaration	[id(154), helpstring("method ApplySignatureValue")] HRESULT ApplySignatureValue([in] BSTR b64SigValXml, [out,retval] BSTR* signedXML);
Java Interface	public String applySignatureValue (String b64SigValXml); String signedXML = sigObj.applySignatureValue (b64SigValXml);
C Interface	ISignature_ApplySignatureValue (ISignature *pSig, BSTR b64SigValXml, BSTR* signedXML);
C++ Interface	pSig->ApplySignatureValue (BSTR b64SigValXml, BSTR* signedXML);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXML = sigObj.ApplySignatureValue (b64SigValXml);
Supported Platforms	All

This method is part of the split signing mechanism supported by SecureXML. ApplySignatureValue should only be called after a call to GetSignedInfoDigest or GetSignedInfoDigestFromByteArray method on the server. The input to this method is the value returned by a call to SignSignedInfoDigest method on the client side. This method applies the signature value to the XML being signed as passed to the previous call to GetSignedInfoDigest or GetSignedInfoDigestFromByteArray methods. The returned value is the complete signed XML.

ApplySignatureValueGetByteArray

IDL File Declaration	[id(156), helpstring("method ApplySignatureValueGetByteArray")] HRESULT ApplySignatureValueGetByteArray([in] BSTR b64SigValXml, [out,retval] VARIANT* signedXmlByteArray);
Java Interface	public byte [] applySignatureValueGetByteArray (String b64SigValXml); byte [] signedXmlByteArray = sigObj.applySignatureValueGetByteArray (b64SigValXml);
C Interface	ISignature_ApplySignatureValueGetByteArray (ISignature *pSig, BSTR b64SigValXml, VARIANT * signedXmlByteArray);
C++ Interface	pSig->ApplySignatureValueGetByteArray (BSTR b64SigValXml, VARIANT * signedXmlByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXmlByteArray = sigObj.ApplySignatureValueGetByteArray (b64SigValXml);
Supported Platforms	All

This method is part of the split signing mechanism supported by SecureXML. ApplySignatureValueGetByteArray should only be called after a call to GetSignedInfoDigest or GetSignedInfoDigestFromByteArray method on the server. The input to this method is the value returned by a call to SignSignedInfoDigest method on the client side. This method applies the signature value to the XML being signed as passed to the previous call to GetSignedInfoDigest or GetSignedInfoDigestFromByteArray methods. The returned value is the complete signed XML.

Base64DecodeBufferToFile

IDL File Declaration	[id(112), helpstring("method Base64DecodeBufferToFile")] HRESULT Base64DecodeBufferToFile([in] BSTR encodedBuffer, [in] BSTR outFilePath, [out,retval] BSTR* decodedFilePath);
Java Interface	public String base64DecodeBufferToFile(String encodedBuffer, String lastParam); String path = sigObj. base64DecodeBufferToFile (encodedBuffer, fileName);
C Interface	ISignature_Base64DecodeBufferToFile (ISignature *pSig, BSTR encodedBuffer, BSTR outFilePath, BSTR* decodedFilePath);
C++ Interface	pSig->Base64DecodeBufferToFile (BSTR encodedBuffer, BSTR outFilePath, BSTR* decodedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	path = sigObj.Base64DecodeBufferToFile (encodedBuffer, filename);
Supported Platforms	All

This method saves the given base64 encoded string to the given filename file after base64 decoding it. If filename is null, it saves it to a temporary file, which is deleted during object destruction. The decodedFilePath contains full path for the saved file.

If invoked from a browser, the input outFilePath parameter is ignored and a temporary file is created instead.

Base64DecodeByteArrayToByteArray

IDL File Declaration	[id(137), helpstring("method Base64DecodeByteArrayToByteArray")] HRESULT Base64DecodeByteArrayToByteArray([in] VARIANT encodedBuffer, [out,retval] VARIANT* decodedBuffer);
Java Interface	Public byte [] base64DecodeByteArrayToByteArray(byte [] encodedBuffer); byte [] decodedBuffer = sigObj.base64DecodeByteArrayToByteArray (encodedBuffer);
C Interface	ISignature_Base64DecodeByteArrayToByteArray (ISignature *pSig, VARIANT encodedBuffer, VARIANT *decodedBuffer);
C++ Interface	pSig->Base64DecodeByteArrayToByteArray (VARIANT encodedBuffer, VARIANT *decodedBuffer);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	decodedBuffer = sigObj.Base64DecodeByetArrayToByteArray (encodedBuffer);
Supported Platforms	All

This method base64 decodes the input byte array and returns the decoded byte array to the calling program. A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

Base64DecodeByteArrayToFile

IDL File Declaration	[id(136), helpstring("method Base64DecodeByteArrayToFile")] HRESULT Base64DecodeByteArrayToFile([in] VARIANT encodedBuffer, [in] BSTR outFilePath, [out,retval] BSTR* decodedFilePath);
Java Interface	public String base64DecodeByteArrayToFile(byte [] encodedBuffer, String outFilePath); String decodedFilePath = sigObj.base64DecodeByteArrayToFile (encodedBuffer, outFilePath);
C Interface	ISignature_Base64DecodeByteArrayToFile (ISignature *pSig, VARIANT encodedBuffer, BSTR outFilePath, BSTR *decodedFilePath);
C++ Interface	pSig->Base64DecodeBufferToFile (VARIANT encodedBuffer, BSTR outFilePath, BSTR *decodedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	decodedFilePath = sigObj.Base64DecodeByteArrayToFile (encodedBuffer, outFilePath);
Supported Platforms	All

This method base64 decodes the input byte array and saves it as outFilePath. If outFilePath is null it is saved as a temporary file, which is deleted during object destruction. On return devcodeFilePath has the full path of the saved file. A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

If invoked from a browser, the input parameter outFilePath is ignored and a temporary file is created instead.

Base64DecodeFileToFile

IDL File Declaration	[id(113), helpstring("method Base64DecodeFileToFile")] HRESULT Base64DecodeFileToFile([in] BSTR encodedFilePath, [in] BSTR outFilePath, [out,retval] BSTR* decodedFilePath);
Java Interface	public String base64DecodeFileToFile(String encodedFilePath, String outFilePath); String decodedFilePath = sigObj.base64DecodeFileToFile (encodedFilePath, outFilePath);
C Interface	ISignature_Base64DecodeFileToFile (ISignature *pSig, BSTR encodedFilePath, BSTR outFilePath, BSTR* decodedFilePath);
C++ Interface	pSig->Base64DecodeFileToFile (BSTR encodedFilePath, BSTR outFilePath, BSTR* decodedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	path = sigObj.Base64DecodeFileToFile (encodedFilePath, outFilePath);
Supported Platforms	All

This method base64 decodes the given base64 encoded file and saves it to outFilePath. If outFilePath is null, it saves it to a temporary file, which is deleted during object destruction. The decodedFilePath contains full path for the saved file.

If invoked from a browser, the input parameter outFilePath is ignored and a temporary file is created instead.

Base64EncodeByteArrayToByteArray

IDL File Declaration	[id(138), helpstring("method Base64EncodeByteArrayToByteArray")] HRESULT Base64EncodeByteArrayToByteArray([in] VARIANT inputBinary, [out,retval] VARIANT* encodedBuffer);
Java Interface	public byte [] base64EncodeByteArrayToByteArray(byte [] inputBinary); byte [] encodedBuffer = sigObj.base64EncodeByteArrayToByteArray (inputBinary);
C Interface	ISignature_Base64EncodeByteArrayToByteArray (ISignature *pSig, VARIANT inputBinary, VARIANT *encodedBuffer);
C++ Interface	pSig->Base64EncodeByteArrayToByteArray (VARIANT inputBinary, VARIANT *encodedBuffer);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	decodedBuffer = sigObj.Base64EncodeByetArrayToByteArray (inputBinary);
Supported Platforms	All

This method base64 encodes the input byte array and returns the encoded byte array to the calling program. A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

Base64EncodeByteArrayToFile

IDL File Declaration	[id(139), helpstring("method Base64EncodeByteArrayToFile")] HRESULT Base64EncodeByteArrayToFile([in] VARIANT inputBinary, [in] BSTR outFilePath, [out,retval] BSTR* encodedFilePath);
Java Interface	public String base64EncodeByteArrayToFile(byte [] inputBinary, String outFilePath); String encodedFilePath = sigObj.base64EncodeByteArrayToFile (inputBinary, outFilePath);
C Interface	ISignature_Base64EncodeByteArrayToFile (ISignature *pSig, VARIANT inputBinary, BSTR outFilePath, BSTR * encodedFilePath);
C++ Interface	pSig->Base64EncodeBufferToFile (VARIANT inputBinary, BSTR outFilePath, BSTR * encodedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	encodedFilePath = sigObj.Base64EncodeByteArrayToFile (inputBinary, outFilePath);
Supported Platforms	All

This method base64 encodes the input byte array and saves it as outFilePath. If outFilePath is null it is saved as a temporary file, which is deleted during object destruction. On return devcodeFilePath has the full path of the saved file. A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

If invoked from a browser, the input parameter outFilePath is ignored and a temporary file is created instead.

Base64EncodeStrToFile

IDL File Declaration	[id(140), helpstring("method Base64EncodeStrToFile")] HRESULT Base64EncodeStrToFile([in] BSTR inputStr, [in] BSTR outFilePath, [out,retval] BSTR *encodedFilePath);
Java Interface	public String base64EncodeStrToFile(String inputStr, String outFilePath); String encodedFilePath = sigObj.base64EncodeStrToFile (inputStr, fileName);
C Interface	ISignature_Base64EncodeStrToFile (ISignature *pSig, BSTR inputStr, BSTR outFilePath, BSTR* encodedFilePath);
C++ Interface	pSig->Base64EncodeStrToFile (BSTR inputStr, BSTR outFilePath, BSTR* encodedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	path = sigObj.Base64EncodeStrToFile (inputStr, outFilePath);
Supported Platforms	All

This method saves the given string to the given filename file after base64 encoding it. If filename is null, it saves it to a temporary file, which is deleted during object destruction. The encodedFilePath contains full path for the saved file.

If invoked from a browser, the input parameter outFilePath is ignored and a temporary file is created instead.

Base64EncodeStrToStr

IDL File Declaration	[id(141), helpstring("method Base64EncodeStrToStr")] HRESULT Base64EncodeStrToStr([in] BSTR inputStr, [out,retval] BSTR* encodedStr);
Java Interface	public String base64EncodeStrToStr(String inputStr); String encodedStr = sigObj.base64EncodeStrToStr (inputStr);
C Interface	ISignature_Base64EncodeStrToStr (ISignature *pSig, BSTR inputStr, BSTR* encodedStr);
C++ Interface	pSig->Base64EncodeStrToStr (BSTR inputStr, BSTR* encodedStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	encodedStr = sigObj.Base64EncodeStrToStr (inputStr);
Supported Platforms	All

This method returns a base64 encoded version of the inputStr.

CaptureLiveSignature

IDL File Declaration	[id(86), helpstring("method CaptureLiveSignature")] HRESULT CaptureLiveSignature([out, retval] BSTR *signatureFilePath);
Java Interface	public String captureLiveSignature(); String signatureFilePath = sigObj.captureLiveSignature();
C Interface	ISignature_CaptureLiveSignature (ISignature *pSig, BOOL * signatureFilePath);
C++ Interface	pSig->CaptureLiveSignature(BSTR * signatureFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signatureFilePath = sigObj.CaptureLiveSignature()
Supported Platforms	Windows Only

It lets users capture live signature from signature pad or mouse without having to create XML Signature. It is useful for applications where all you need is an image and a PKI based XML Signature is not required. The returned value is the full path to the PNG file just captured. Please note that this file is automatically deleted during the object destruction process. So if you would like to have access to this file after SecureXML Digital Signature object instance no longer exists, please copy or move this file to a known location other than the one returned by signatureFilePath.

ChangeOrAddProperty

IDL File Declaration	[id(84), helpstring("method ChangeOrAddProperty")] HRESULT ChangeOrAddProperty([in] BSTR propertyName, [in]BSTR propertyValue);
Java Interface	public void changeOrAddProperty(String propertyName, String propertyValue); sigObj.changeOrAddProperty (“Location”, “Honolulu”);
C Interface	ISignature_ChangeOrAddProperty (ISignature *pSig, BSTR propertyName, BSTR propertyValue);
C++ Interface	pSig->ChangeOrAddProperty (BSTR propertyName, BSTR propertyValue);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ChangeOrAddProperty (“Location”, “Honolulu”) The parenthesis are not allowed when invoking from VBScript
Supported Platforms	All

This method takes the pain out of keeping count of currently set signature properties. All you need to provide is a property name and its value and this method takes care of the rest. It looks for an existing property with the matching name, if found, it sets the value of the property to the new value. If a matching property is not found it adds the given property and increments the PropertyCount.

CoSignFile

IDL File Declaration	[id(90), helpstring("method CoSignFile")] HRESULT CoSignFile([in] BSTR inputSignedXMLFile, [in] BSTR outFileName, [out, retval] BSTR *outFilePath);
Java Interface	public String coSignFile(String inputSignedXMLFile, String outFileName); String outFilePath = sigObj.coSignFile ("C:\\temp\\contract.xml", "C:\\temp\\CoSignedContract.xml);
C Interface	ISignature_CoSignFile (ISignature *pSig, BSTR inputSignedXMLFile, BSTR outFileName, BSTR *outFilePath);
C++ Interface	pSig->CoSignFile (BSTR inputSignedXMLFile, BSTR outFileName, BSTR *outFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	outFilePath = sigObj.CoSignFile ("C:\\temp\\contract.xml", "C:\\temp\\CoSignedContract.xml);
Supported Platforms	All

This mehod is used to cosign existing signed data in the input XML file. The input signed XML should be created using SignFile, SignFiles or SignDataStr methods. If outFileName is null, the cosigned XML is saved in a temporary file and the full path to this temporary file is returned as outFilePath. If outFileName is non-null, outFilePath points to outFileName.

CoSignXMLStr

IDL File Declaration	[id(89), helpstring("method CoSignXMLStr")] HRESULT CoSignXMLStr([in] BSTR signedDataXMLStr, [out, retval] BSTR *coSignedXMLStr);
Java Interface	public String coSignXMLStr(String signedDataXMLStr); String coSignedXMLStr = sigObj.coSignXMLStr (signedDataXMLStr);
C Interface	ISignature_CoSignXMLStr (ISignature *pSig, BSTR signedDataXMLStr, BSTR *coSignedXMLStr);
C++ Interface	pSig->CoSignXMLStr (BSTR signedDataXMLStr, BSTR *coSignedXMLStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	coSignedXMLStr = sigObj.CoSignXMLStr (signedDataXMLStr);
Supported Platforms	All

This mehod is used to cosign existing signed data in the input XML string. The input signed XML should be created using SignFile, SignFiles or SignDataStr methods.

DecryptFile

IDL File Declaration	[id(96), helpstring("method DecryptFile")] HRESULT DecryptFile([in] BSTR inputFile, [in] BSTR outputFile, [out,retval] BSTR* decryptedFile);
Java Interface	public String decryptFile(String inputFile, String lastParam); String decryptedFilePath = sigObj.decryptFile (inputFile, outputFile);
C Interface	ISignature_DecryptFile (ISignature *pSig, BSTR inputFile, BSTR outputFile, BSTR *decryptedFile);
C++ Interface	pSig->DecryptFile(BSTR inputFile, BSTR outputFile, BSTR *decryptedFile);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	decryptedFile = sigObj.DecryptFile (inputFile, outputFile);
Supported Platforms	Windows Only

This method is used to decrypt existing encrypted data in inputFile. If outputFile is null, the decrypted file is created in the current user's temporary folder. The return value "decryptedFile" is the full path of the decrypted file. The current user's "MY" certificate store must contain a certificate with matching private key in order for this method to succeed.

DecryptStr

IDL File Declaration	[id(94), helpstring("method DecryptStr")] HRESULT DecryptStr([in] BSTR cipherText, [out,retval] BSTR* plainText);
Java Interface	public String decryptStr(String cipherText); String plainText = sigObj.decryptStr (cipherText);
C Interface	ISignature_DecryptStr (ISignature *pSig, BSTR cipherText, BSTR * plainText);
C++ Interface	pSig->DecryptStr(BSTR cipherText, BSTR * plainText);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	plainText = sigObj.DecryptStr (cipherText);
Supported Platforms	Windows Only

This method is used to decrypt existing encrypted data in cipherText buffer. The current user's "MY" certificate store must contain a certificate with matching private key in order for this method to succeed.

DeleteSignatureFromFile

IDL File Declaration	[id(119), helpstring("method DeleteSignatureFromFile")] HRESULT DeleteSignatureFromFile([in] BSTR signedXMLFile, [in] BSTR sigId, [in] BSTR outFilePath, [out,retval] BSTR* newSigFilePath);
Java Interface	public String deleteSignatureFromFile(String signedXMLFile, String sigId, String outFilePath) String outFilePath = sigObj.deleteSignatureFromFile (signedXMLFile, sigId);
C Interface	ISignature_DeleteSignatureFromFile (ISignature *pSig, BSTR signedXMLFile, BSTR sigId, BSTR outFilePath, BSTR* newSigFilePath);
C++ Interface	pSig->DeleteSignatureFromFile (BSTR signedXMLFile, BSTR sigId, BSTR outFilePath, BSTR* newSigFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	outFilePath = sigObj.DeleteSignatureFromFile (signedXMLFile, sigId);
Supported Platforms	All

This method deletes the signature element with matching signature id from the given signature file. If the signature file contains base64 encoded signed XML, and Base64DecodeXML = 1, then a base64 decode is performed before attempting to parse the input file and after deleting the signature element, the resulting XML is base64 encoded before being saved in the outFilePath. If outFilePath is NULL, the result is saved in a temporary file. Upon return newSigFilePath contains the full path of the saved file.

DeleteSignatureFromXMLStr

IDL File Declaration	[id(118), helpstring("method DeleteSignatureFromXMLStr")] HRESULT DeleteSignatureFromXMLStr([in] BSTR signedXMLStr, [in] BSTR sigId, [out,retval] BSTR* newSigXMLStr);
Java Interface	public String deleteSignatureFromXMLStr(String signedXMLStr, String sigId); String newSigXMLStr = sigObj.deleteSignatureFromXMLStr (signedXMLStr, sigId);
C Interface	ISignature_DeleteSignatureFromXMLStr (ISignature *pSig, BSTR signedXMLStr, BSTR sigId, BSTR* newSigXMLStr);
C++ Interface	pSig->DeleteSignatureFromXMLStr (BSTR signedXMLStr, BSTR sigId, BSTR* newSigXMLStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	newSigXMLStr = sigObj.DeleteSignatureFromXMLStr (signedXMLStr, sigId);
Supported Platforms	All

This method deletes the signature element with matching signature id from the given signed XML. If the input contains base64 encoded signed XML, and Base64DecodeXML = 1, then a base64 decode is performed before attempting to parse the input and after deleting the signature element, the resulting XML is base64 encoded before being returned to the calling program.

EncryptFile

IDL File Declaration	[id(95), helpstring("method EncryptFile")] HRESULT EncryptFile([in] BSTR inputFile, [in] BSTR outputFile, [out,retval] BSTR* encryptedFile);
Java Interface	public String encryptFile(String inputFile, String lastParam); String encryptedFilePath = sigObj.encryptFile (inputFile, outputFile);
C Interface	ISignature_EncryptFile (ISignature *pSig, BSTR inputFile, BSTR outputFile, BSTR *encryptedFile);
C++ Interface	pSig->EncryptFile(BSTR inputFile, BSTR outputFile, BSTR *encryptedFile);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	encryptedFile = sigObj.EncryptFile (inputFile, outputFile);
Supported Platforms	Windows Only

This method is used to encrypt existing data in inputFile. If outputFile is null, the encrypted file is created in the current user's temporary folder. The return value "encryptedFile" is the full path of the encrypted file. If neither RecipientCertificates nor RecipientCertificateFiles is set, a certificate selection dialog box is displayed. The certificates shown are from the current user's "addressbook" store unless RecipientCertificateStore property to set to a different store.

EncryptStr

IDL File Declaration	[id(93), helpstring("method EncryptStr")] HRESULT EncryptStr([in] BSTR inputStr, [out,retval] BSTR* cipherStr);
Java Interface	public String encryptStr(String lastParam) String encryptedStr = sigObj.encryptStr (signedDataXMLStr);
C Interface	ISignature_EncryptStr (ISignature *pSig, BSTR signedDataXMLStr, BSTR *encryptedStr);
C++ Interface	pSig->EncryptStr (BSTR signedDataXMLStr, BSTR *encryptedStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	encryptedStr = sigObj.EncryptStr (signedDataXMLStr);
Supported Platforms	Windows Only

This method is used to encrypt existing signed data in the input XML string. The input is a null terminated string. If neither RecipientCertificates nor RecipientCertificateFiles is set, a certificate selection dialog box is displayed. The certificates shown are from the current user's "addressbook" store unless RecipientCertificateStore property to set to a different store.

GetCertificateInfo

IDL File Declaration	[id(27), helpstring("method GetCertificateInfo")] HRESULT GetCertificateInfo([in] long index, [in] long valIndex, [out, retval] BSTR *value);
Java Interface	public String getCertificateInfo(int index, int valIndex); String certSubject = sigObj.getCertificateInfo (0, 3);
C Interface	ISignature_GetCertificateInfo (ISignature *pSig, long index, long valIndex, BSTR *value);
C++ Interface	pSig->GetCertificateInfo (long index, long valIndex, BSTR *value);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	certSubject = sigObj.GetCertificateInfo (0,3)
Supported Platforms	All

Use this method to programmatically access the information about various certificates installed in your local machine. By default it accesses the "MY" store of the current user. You can also call SetStoreName method prior to calling GetCertificateInfo to access any other user defined certificate store. The index parameter can vary from 0 to (CertificateCount -1). It can be -2 if the active signer certificate is PFX file based or it can be -3 if the active signer certificate is from local windows certificate store.

Depending on the value of the valIndex parameter it returns various values associated with a certificate corresponding to the index. valIndex has the following meaning:

valIndex Value	What the *value returns
1	Certificate Serial Number
2	Certificate Issuer Name
3	Certificate Subject Name
4	Certificate Expiration Date
5	Certificate Subject Short or Friendly Name

GetError

IDL File Declaration	[id(19), helpstring("method GetError")] HRESULT GetError([out, retval] BSTR *errorString);
Java Interface	public String getError(); String errorString = sigObj.getError ();
C Interface	ISignature_GetError (ISignature *pSig, BSTR *errorString);
C++ Interface	pSig->GetEror (BSTR *errorString);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	errorString = sigObj.GetError ()
Supported Platforms	All

It returns the string containing the stack of all the errors the previous call into the SecureXML component may have encountered. Please note that the error stack may also contain errors prior to the previous method/property invocation.

GetErrorDetail

IDL File Declaration	[id(147), helpstring("method GetErrorDetail")] HRESULT GetErrorDetail([in] LONG errorNum, [out,retval] BSTR* errorDesc);
Java Interface	public String getErrorDetail(int erronNum); String errorString = sigObj.getErrorDetail (errorNum);
C Interface	ISignature_GetErrorDetail (ISignature *pSig, LONG errorNum, BSTR *errorString);
C++ Interface	pSig->GetEror (LONG errorNum, BSTR *errorString);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	errorString = sigObj.GetErrorDetail (errorNum)
Supported Platforms	All

It returns the string containing the description corresponding to the error number passed.

GetLastError

IDL File Declaration	[id(23), helpstring("method GetLastError")] HRESULT GetLastError([out, retval] long *errorNum);
Java Interface	public int getLastError(); int errorNum = sigObj.getLastError ();
C Interface	ISignature_GetLastError (ISignature *pSig, long *errorNum);
C++ Interface	pSig->GetLastEror (long *errorNum);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	errorNum = sigObj.GetLastError ()
Supported Platforms	All

Returns the last error number. Please refer to the error code list at the end of this manual for error descriptions. Please note, if the previous call returned a valid result, the value returned by GetLastError() is not relevant.

GetSignedDocumentB64Str

IDL File Declaration	[id(157), helpstring("method GetSignedDocumentB64Str")] HRESULT GetSignedDocumentB64Str([in] LONG sigIndex, [in] LONG uriIndex, [out,retval] BSTR* signedDocB64Str);
Java Interface	public String getSignedDocumentB64Str(int sigIndex, int uriIndex); String signedDocB64Str = sigObj.getSignedDocumentB64Str (0, 0);
C Interface	ISignature_GetSignedDocumentB64Str (ISignature *pSig, LONG sigIndex, LONG uriIndex, BSTR * signedDocB64Str);
C++ Interface	pSig->GetSignedDocumentB64Str (LONG sigIndex, LONG uriIndex, BSTR * signedDocB64Str);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedDocB64Str = sigObj.GetSignedDocumentB64Str (sigIndex, uriIndex)
Supported Platforms	All

A call to GetSignedDocumentB64Str is valid only after a call to one of the signature verification methods.

Returns the base64 encoded string containing the signed document as indicated by the <Reference> child element of the <Signature> element corresponding to the given sigIndex, uriIndex pair.

sigIndex can take values from 0 to SignatureCount – 1
uriIndex can take values from 0 to SignedDocumentCount(sigIndex) - 1

GetSignedDocumentByteArray

IDL File Declaration	[id(158), helpstring("method GetSignedDocumentByteArray")] HRESULT GetSignedDocumentByteArray([in] LONG sigIndex, [in] LONG uriIndex, [out,retval] VARIANT* signedDocumentByteArray);
Java Interface	public byte [] getSignedDocumentByteArray(int sigIndex, int uriIndex); byte [] signedDocumentByteArray = sigObj.getSignedDocumentByteArray (0, 0);
C Interface	ISignature_GetSignedDocumentByteArray (ISignature *pSig, LONG sigIndex, LONG uriIndex, VARIANT* signedDocumentByteArray);
C++ Interface	pSig->GetSignedDocumentByteArray (LONG sigIndex, LONG uriIndex, VARIANT* signedDocumentByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedDocumentByteArray = sigObj.GetSignedDocumentByteArray (sigIndex, uriIndex)
Supported Platforms	All

A call to GetSignedDocumentByteArray is valid only after a call to one of the signature verification methods.

Returns the byte array containing the signed document as indicated by the <Reference> child element of the <Signature> element corresponding to the given sigIndex, uriIndex pair.

sigIndex can take values from 0 to SignatureCount – 1
uriIndex can take values from 0 to SignedDocumentCount(sigIndex) - 1

GetSignedInfoDigest

IDL File Declaration	[id(149), helpstring("method GetSignedInfoDigest")] HRESULT GetSignedInfoDigest([in] BSTR xmlStr, [in] BSTR signatureId, [out,retval] BSTR* signedInfoDigest);
Java Interface	public String getSignedInfoDigest(String xmlStr, String signatureId); String signedInfoDigest = sigObj.getSignedInfoDigest (xmlStr, signatureId);
C Interface	ISignature_GetSignedInfoDigest (ISignature *pSig, BSTR xmlStr, BSTR signatureId, BSTR * signedInfoDigest);
C++ Interface	pSig->GetSignedInfoDigest (BSTR xmlStr, BSTR signatureId, BSTR * signedInfoDigest);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedInfoDigest = sigObj.GetSignedInfoDigest (xmlStr, signatureId)
Supported Platforms	All

This method is used for split signing operation. Typically this method is used on server side signed info digest calculation. It performs all the steps needed to sign the xmlStr except it stops just before the final signature value calculation and returns the digest value to the calling program. The calling program can take this digest value to the client side, invoke SignSignedInfoDigest method and obtain the signature value which would then need to be sent back to server side. The server side code would then invoke ApplySignatureValue method to complete the signing process. The input parameters to GetSignedInfoDigest are xmlStr (it can be null if either AttachedObjects or DetachedObjects property is set), the XML template to be signed and signatureId, the Id of the signature element to populate or create.

If SignerCertificateChain is not set before calling GetSignedInfoDigesMethod, the XML template parameter (xmlStr) must contain the <SignatureMethod Algorithm="…" /> element present in the <SignedInfo> section of the <Signature> element to be processed by this method. Otherwise an error is reported.

GetSignedInfoDigestFromByteArray

IDL File Declaration	[id(155), helpstring("method GetSignedInfoDigestFromByteArray")] HRESULT GetSignedInfoDigestFromByteArray([in] VARIANT xmlByteArray, [in] BSTR signatureId, [out,retval] BSTR* signedInfoDigest);
Java Interface	public String getSignedInfoDigestFromByteArray(byte [] xmlByteArray, String signatureId); String signedInfoDigest = sigObj.getSignedInfoDigestFromByteArray (xmlByteArray, signatureId);
C Interface	ISignature_GetSignedInfoDigestFromByteArray (ISignature *pSig, BSTR xmlByteArray, BSTR signatureId, BSTR * signedInfoDigest);
C++ Interface	pSig->GetSignedInfoDigestFromByteArray (BSTR xmlByteArray, BSTR signatureId, BSTR * signedInfoDigest);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedInfoDigest = sigObj.GetSignedInfoDigestFromByteArray (xmlByteArray, signatureId)
Supported Platforms	All

This method is used for split signing operation. Typically this method is used on server side signed info digest calculation. It performs all the steps needed to sign the xmlByteArray except it stops just before the final signature value calculation and returns the digest value to the calling program. The calling program can take this digest value to the client side, invoke SignSignedInfoDigest method and obtain the signature value which would then need to be sent back to server side. The server side code would then invoke ApplySignatureValue method to complete the signing process. The input parameters to GetSignedInfoDigest are xmlByteArray (it can be null if either AttachedObjects or DetachedObjects property is set), the XML template to be signed and signatureId, the Id of the signature element to populate or create.

If SignerCertificateChain is not set before calling GetSignedInfoDigesMethodFromByteArray, the XML template parameter (xmlByteArray) must contain the <SignatureMethod Algorithm="…" /> element present in the <SignedInfo> section of the <Signature> element to be processed by this method. Otherwise an error is reported.

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

GetSignedFileObject

IDL File Declaration	[id(48), helpstring("method GetSignedFileObject")] HRESULT GetSignedFileObject([in] BSTR signedXML, [in] BSTR saveDir, [out, retval] BSTR *signedFilePath);
Java Interface	public String getSignedFileObject(String signedXML, String saveDir); String signedFilePath = sigObj.getSignedFileObject (signedXML, saveDir);
C Interface	ISignature_GetSignedFileObject (ISignature *pSig, BSTR signedXML, BSTR saveDir, BSTR *signedFilePath);
C++ Interface	pSig->GetSignedFileObject (BSTR signedXML, BSTR saveDir, BSTR *signedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedFilePath = sigObj.GetSignedFileObject (signedXML, saveDir)
Supported Platforms	All

This method is provided to retrieve the file which was signed using ***SignFile*** method. It does the base64 decode of the file data contained in the signed XML and stores it either in the path pointed to by ***saveDir*** parameter or if ***saveDir*** is null, in the current user's temporary directory with the original file name. The ***signedFilePath*** contains the complete path for the extracted signed file. If the operation was not successful, ***signedFilePath*** is set to null.

GetSigPropValueByName

IDL File Declaration	[id(111), helpstring("method GetSigPropValueByName")] HRESULT GetSigPropValueByName([in] LONG sigIndex, [in] BSTR propName, [out,retval] BSTR* propValue);
Java Interface	public String getSigPropValueByName(int sigIndex, String lastParam); String propValue = sigObj.getSigPropValueByName (0,"CamValidationResponse"); // Returns the value of the CamValidationResponse property for the 0th signature in the most recently verified signature.
C Interface	ISignature_GetSigPropValueByName (ISignature *pSig, LONG sigIndex, BSTR propName, BSTR *propValue);
C++ Interface	pSig->GetSigPropValueByName (LONG sigIndex, BSTR propName, BSTR *propValue);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	propValue = sigObj. GetSigPropValueByName (sigIndex, propName)
Supported Platforms	All

This method is valid only after a call to Verify/VerifyXMLStr/SecureXMLVerify methods. It returns the value of the "propName" signature property for the signature referred to by "sigIndex".

sigIndex can vary between 0 and (SignatureCount – 1).

GetVersion

IDL File Declaration	[id(52), helpstring("method GetVersion")] HRESULT GetVersion([out, retval] BSTR *version);
Java Interface	public String getVersion(); String secureXMLVersion = sigObj.getVersion ();
C Interface	ISignature_GetVersion (ISignature *pSig, BSTR * secureXMLVersion);
C++ Interface	pSig->GetVersion (BSTR * secureXMLVersion);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	secureXMLVersion = sigObj.GetVersion ()
Supported Platforms	All

It returns the string containing version number of SecureXML object being used.

GetX509Certificate

IDL File Declaration	[id(24), helpstring("method GetX509Certificate")] HRESULT GetX509Certificate([in] BSTR certID, [out, retval] BSTR *certData);
Java Interface	public String getX509Certificate(String certID); String certData = sigObj.getX509Certificate (certID);
C Interface	ISignature_GetX509Certificate (ISignature *pSig, BSTR certID, BSTR *certData);
C++ Interface	pSig->GetX509Certificate (BSTR certID, BSTR *certData);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	certData = sigObj.GetX509Certificate (certID)
Supported Platforms	All

Returns the base64 encoded X509 certificate data corresponding to the given certificate serial number (cerID). The certificate must be in the current user's "MY" store or in the store set by SetStoreName mehod.

GetX509CertificateChain

IDL File Declaration	[id(150), helpstring("method GetX509CertificateChain")] HRESULT GetX509CertificateChain([in] BSTR certID, [out,retval] BSTR* certChainAsB64XmlStr);
Java Interface	public String getX509CertificateChain(String certID); String certChainAsB64XmlStr = sigObj.getX509CertificateChain (certID);
C Interface	ISignature_GetX509CertificateChain (ISignature *pSig, BSTR certID, BSTR * certChainAsB64XmlStr);
C++ Interface	pSig->GetX509CertificateChain (BSTR certID, BSTR * certChainAsB64XmlStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	certChainAsB64XmlStr = sigObj.GetX509CertificateChain (certID)
Supported Platforms	All

Returns the base64 encoded XML containing base64 encoded X509 certificate chain data corresponding to the given certificate serial number (cerID). The certificate must be in the current user's "MY" store or in the store set by SetStoreName mehod. This method is useful in split signing application where the signed info digest calculation is performed on the server while the signature value calculation if performed on the client side. You would call GetX509CertificateChain on the client side and pass the certChainAsB64XmlStr value to the server side. On the server side you can now invoke SetSignerCertificateChain with this data as input before invoking GetSignedInfoDigest or GetSignedInfoDigestFromByteArray methods.

GunZipFile

IDL File Declaration	[id(114), helpstring("method GunZipFile")] HRESULT GunZipFile([in] BSTR gZippedFile, [in] BSTR gUnZippedFile, [out,retval] BSTR* gUnZippedFilePath);
Java Interface	public String gunZipFile(String gZippedFile, String gUnZippedFile); String gUnZippedFilePath = sigObj.gunZipFile (gZippedFile, gUnZippedFile);
C Interface	ISignature_GunZipFile (ISignature *pSig, BSTR gZippedFile, BSTR gUnZippedFile, BSTR* gUnZippedFilePath);
C++ Interface	pSig->GunZipFile (BSTR gZippedFile, BSTR gUnZippedFile, BSTR* gUnZippedFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	gUnZippedFilePath = sigObj.GunZipFile (gZippedFile, gUnZippedFile)
Supported Platforms	All

It gunzips the gZippedFile and saves the uncompressed data as gUnZippedFile. If gUnZippedFile is NULL, the uncompressed data is saved as a temp file. gUnZippedFilePath is set to the full path of the uncompressed file.

PFXExportActiveCertificate

IDL File Declaration	[id(59), helpstring("method PFXExportActiveCertificate")] HRESULT PFXExportActiveCertificate([in] BSTR password, [out, retval] BSTR *pfxFilePath);
Java Interface	public String pFXExportActiveCertificate(String password); String pfxFilePath = sigObj.pFXExportActiveCertificate (password);
C Interface	ISignature_PFXExportActiveCertificate (ISignature *pSig, BSTR password, BSTR *pfxFilePath);
C++ Interface	pSig->PFXExportActiveCertificate (BSTR password, BSTR *pfxFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pfxFilePath = sigObj.PFXExportActiveCertificate (password)
Supported Platforms	All

PFXExportActiveCertificate saves you the trouble of providing a certificate serial number. Once the user has selected a certificate for signing, you can export that particular certificate by supplying a password parameter. Internally this function takes the certificate serial number of the currently active certificate and calls PFXExportCertificate. The return value is again file path for temp file where the exported certificate has been stored.

PFXExportCertificate

IDL File Declaration	[id(21), helpstring("method PFXExportCertificate")] HRESULT PFXExportCertificate([in] BSTR certID, [in] BSTR password, [out, retval] BSTR *pfxFilePath);
Java Interface	public String pFXExportCertificate(String certID, String password); String pfxFilePath = sigObj.pFXExportCertificate (certID, password);
C Interface	ISignature_PFXExportCertificate (ISignature *pSig, BSTR certID, BSTR password, BSTR *pfxFilePath);
C++ Interface	pSig->PFXExportCertificate (BSTR certID, BSTR password, BSTR *pfxFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pfxFilePath = sigObj.PFXExportCertificate (certID, password)
Supported Platforms	All

PFXExportCertificate takes the certificate serial number (certId) and a password as input parameter and returns a file path for a temp file where the exported certificate has been stored. You can use GetCertificateInfo() method to access the certificate serial number of any certificate in the local machine store.

ReadAll

IDL File Declaration	[id(41), helpstring("method ReadAll")] HRESULT ReadAll([in] BSTR fileName, [out, retval] BSTR *fileDataStr);
Java Interface	public String readAll(String lastParam) String filename = "C:\temp\myfile.txt" String fileDataStr = sigObj.readAll (fileName);
C Interface	ISignature_ReadAll (ISignature *pSig, BSTR fileName, BSTR *fileDataStr);
C++ Interface	pSig->ReadAll (BSTR filename, BSTR *fileDataStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	fileDataStr = sigObj.ReadAll (fileName);
Supported Platforms	All

This method is used to read the entire content of a text file. This file could either be local or on the internet. Currently it supports http and ldap access mechanisms. For reading binary data, please use either ReadAllBase64 or ReadAllByteArray method.

ReadAllBase64

IDL File Declaration	[id(101), helpstring("method ReadAllBase64")] HRESULT ReadAllBase64([in] BSTR uri, [out,retval] BSTR* base64EncodedData);
Java Interface	public String readAllBase64(String lastParam) String filename = "C:\temp\myfile.bin" String base64EncodedData = sigObj.readAllBase64 (fileName);
C Interface	ISignature_ReadAllBase64 (ISignature *pSig, BSTR fileName, BSTR * base64EncodedData);
C++ Interface	pSig->ReadAllBase64 (BSTR fileName, BSTR * base64EncodedData);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	base64EncodedData = sigObj.ReadAllBase64 (fileName);
Supported Platforms	All

This method is used to read the entire content of a binary file. The output is a base64 encoded string. This file could either be local or on the internet. Currently it supports http and ldap access mechanisms.

ReadAllByteArray

IDL File Declaration	[id(131), helpstring("method ReadAllByteArray")] HRESULT ReadAllByteArray([in] BSTR fileName, [out,retval] VARIANT* fileDataByteArray);
Java Interface	public byte [] readAllByteArray(String fileName) String fileName = "C:\\temp\\myfile.txt" byte [] fileDataByteArray = sigObj.readAllByteArray (fileName);
C Interface	ISignature_ReadAllByteArray (ISignature *pSig, BSTR fileName, VARIANT * fileDataByteArray);
C++ Interface	pSig->ReadAllByteArray (BSTR filename, VARIANT * fileDataByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	fileDataByteArray = sigObj.ReadAllByteArray (fileName);
Supported Platforms	All

This method is used to read the entire content of a binary file. This file could either be local or on the internet. Currently it supports http and ldap access mechanisms.

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

SaveXMLByteArray

IDL File Declaration	[id(130), helpstring("method SaveXMLByteArray")] HRESULT SaveXMLByteArray([in] VARIANT inputXmlByteArray, [in] BSTR fileName, [out,retval] BSTR* path);
Java Interface	public String saveXMLByteArray(byte [] inputXmlByteArray, String fileName); String path = sigObj.saveXMLByteArray (inputXmlByteArray, fileName);
C Interface	ISignature_SaveXMLByteArray (ISignature *pSig, VARIANT inputXmlByteArray, BSTR fileName, BSTR *path);
C++ Interface	pSig->SaveXMLByteArray (VARIANT inputXmlByteArray, BSTR fileName, BSTR *path);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	path = sigObj.SaveXMLByteArray (inputXmlByteArray, filename);
Supported Platforms	All

This method saves the given inputXmlByteArray byte array to fileName file. If fileName is null, it saves the inputXmlByteArray to a temporary file, which is deleted during object destruction. The path contains full path for the saved file.

If invoked from a browser, the input parameter fileName is ignored and a temporary file is created instead.

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

SaveXMLSignature

IDL File Declaration	[id(17), helpstring("method SaveXMLSignature")] HRESULT SaveXMLSignature([in] BSTR sigFileName);
Java Interface	public void saveXMLSignature(String sigFileName); sigObj.saveXMLSignature (sigFileName);
C Interface	ISignature_SaveXMLSignature (ISignature *pSig, BSTR sigFileName);
C++ Interface	pSig->SaveXMLSignature (BSTR sigFileName);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SaveXMLSignature (sigFileName)
Supported Platforms	All

This moves the temporary file containing the XML Signature produced during a call to Sign method to the file path indicated by sigFileName. If OverwriteFile is set to 1, the destination file, if it already exists, is overwritten. The default behavior is not to overwrite.

SaveXMLStr

IDL File Declaration	[id(42), helpstring("method SaveXMLStr")] HRESULT SaveXMLStr([in] BSTR inputXMLStr, [in] BSTR fileName, [out, retval] BSTR *path);
Java Interface	public String saveXMLStr(String inputXMLStr, String fileName); String path = sigObj.saveXMLStr (inputXMLStr, fileName);
C Interface	ISignature_SaveXMLStr (ISignature *pSig, BSTR inputXMLStr, BSTR fileName, BSTR *path);
C++ Interface	pSig->SaveXMLStr (BSTR inputXMLStr, BSTR fileName, BSTR *path);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	path = sigObj.SaveXMLStr (inputXMLStr, filename);
Supported Platforms	All

This method saves the given inputXMLStr String to filename file. If filename is null, it saves the inputXMLStr to a temporary file, which is deleted during object destruction. The path contains full path for the saved file.

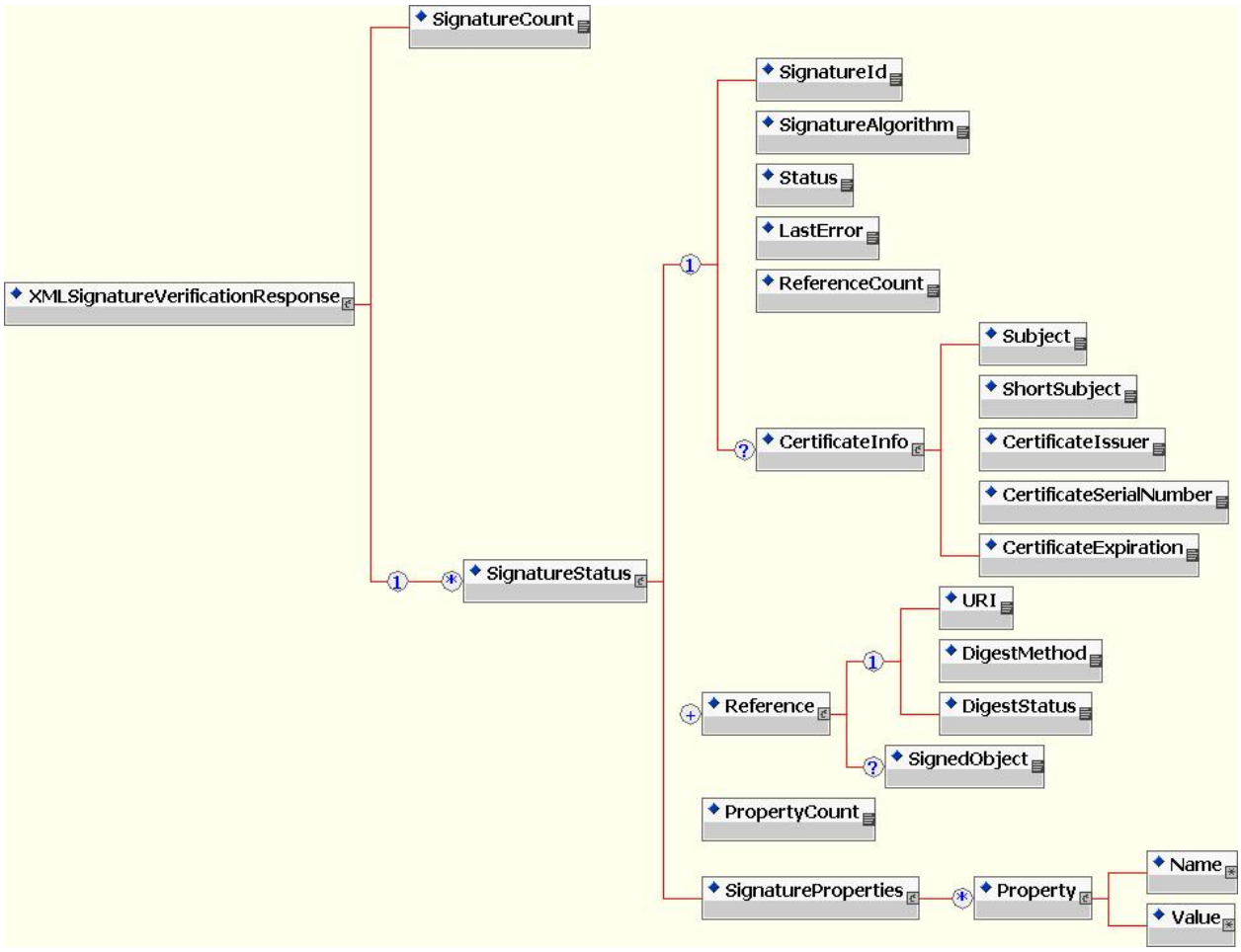
If invoked from a browser, the input parameter fileName is ignored and a temporary file is created instead.

SecureXMLVerify

IDL File Declaration	[id(44), helpstring("method SecureXMLVerify")] HRESULT SecureXMLVerify([in] BSTR signedXML, [out, retval] BSTR *verificationResponse);
Java Interface	public String secureXMLVerify(String signedXML); String verificationResponse = sigObj.secureXMLVerify (signedXML);
C Interface	ISignature_SecureXMLVerify (ISignature *pSig, BSTR signedXML, BSTR *verificationResponse);
C++ Interface	pSig->SecureXMLVerify (BSTR signedXML, BSTR *verificationResponse);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	verificationResponse = sigObj.SecureXMLVerify (signedXML)
Supported Platforms	All

It is recommended that if you are using SecureXMLVerify () method for verfying signature, you set DetailedVerificationFlag to zero since you will always get the detailed information in the response. This flag becomes useful for SecureXMLVerify() method if you have a detached reference which could not be resolved during verification and you need to specify the failed URI's object locations by setting FailedUriFullPath object property.

The input string contains an XML document with one or more XML signatures. The output string provides a the results of the signature verification as per the following content model diagram:



The above diagram corresponds to the following DTD:

```
<?xml encoding='UTF-8' ?>

<!ELEMENT XMLSignatureVerificationResponse (SignatureCount , (SignatureStatus*))>
<!ELEMENT SignatureStatus ((SignatureId , SignatureAlgorithm , Status , LastError , ReferenceCount , CertificateInfo?) ,
Reference+ , PropertyCount , SignatureProperties)>
<!ELEMENT SignatureId (#PCDATA)>
<!ELEMENT SignatureAlgorithm (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
<!ELEMENT LastError (#PCDATA)>
<!ELEMENT ReferenceCount (#PCDATA)>
<!ELEMENT Reference ((URI , DigestMethod , DigestStatus) , SignedObject?)>
<!ELEMENT URI (#PCDATA)>
<!ELEMENT DigestMethod (#PCDATA)>
<!ELEMENT DigestStatus (#PCDATA)>
<!ELEMENT SignedObject (#PCDATA)>
<!ATTLIST SignedObject Encoding CDATA #IMPLIED >
<!ELEMENT SignatureCount (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT CertificateInfo (Subject , ShortSubject , CertificateIssuer , CertificateSerialNumber , CertificateExpiration)>
<!ELEMENT CertificateIssuer (#PCDATA)>
<!ELEMENT CertificateExpiration (#PCDATA)>
<!ELEMENT CertificateSerialNumber (#PCDATA)>
<!ELEMENT PropertyCount (#PCDATA)>
<!ELEMENT SignatureProperties (Property*)>
```

```
<!ELEMENT Name ANY>
<!ELEMENT Value ANY>
<!ELEMENT Property (Name , Value)>
<!ELEMENT ShortSubject (#PCDATA)>
```

SecureXMLVerifyByteArray

IDL File Declaration	[id(133), helpstring("method SecureXMLVerifyByteArray")] HRESULT SecureXMLVerifyByteArray([in] VARIANT signedXmlByteArray, [out,retval] VARIANT* verificationResponseByteArray);
Java Interface	public byte [] secureXMLVerifyByteArray(byte [] signedXmlByteArray); byte [] verificationResponseByteArray = sigObj.secureXMLVerifyByteArray (signedXmlByteArray);
C Interface	ISignature_SecureXMLVerifyByteArrayByteArray (ISignature *pSig, VARIANT signedXmlByteArray, VARIANT * verificationResponseByteArray);
C++ Interface	pSig->SecureXMLVerifyByteArray (VARIANT signedXmlByteArray, VARIANT * verificationResponseByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	verificationResponseByteArray = sigObj.SecureXMLVerifyByteArray (signedXML)
Supported Platforms	All

It is recommended that if you are using SecureXMLVerifyByteArray () method for verfying signature, you set DetailedVerificationFlag to zero since you will always get the detailed information in the response. This flag becomes useful for SecureXMLVerifyByteArray() method if you have a detached reference which could not be resolved during verification and you need to specify the failed URI's object locations by setting FailedUriFullPath object property.

The input byte array contains an XML document with one or more XML signatures. The output byte array provides the results of the signature verification as per the content model diagram described in SecureXMLVerify method description.

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

SecureXMLVerifyFileToBuffer

IDL File Declaration	[id(116), helpstring("method SecureXMLVerifyFileToBuffer")] HRESULT SecureXMLVerifyFileToBuffer([in] BSTR signedXMLFile, [out,retval] BSTR* verificationResponse);
Java Interface	public String secureXMLVerifyFileToBuffer(String signedXMLFile); String verificationResponse = sigObj.secureXMLVerifyFileToBuffer (signedXMLFile);
C Interface	ISignature_SecureXMLVerifyFileToBuffer (ISignature *pSig, BSTR signedXMLFile, BSTR *verificationResponse);
C++ Interface	pSig->SecureXMLVerifyToBuffer (BSTR signedXMLFile, BSTR *verificationResponse);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	verificationResponse = sigObj.SecureXMLVerifyFileToBuffer (signedXMLFile)
Supported Platforms	All

For detailed description please refer to SecureXMLVerify() method. SecureXMLVerifyFileToBuffer() method is a variation where the input is taken from a file instead of a memory buffer.

SecureXMLVerifyFileToFile

IDL File Declaration	[id(117), helpstring("method SecureXMLVerifyFileToFile")] HRESULT SecureXMLVerifyFileToFile([in] BSTR signedXMLFile, [in] BSTR outFilePath, [out,retval] BSTR* verificationResponseFilePath);
Java Interface	public String secureXMLVerifyFileToFile(String signedXMLFile, outFilePath); String verificationResponseFilePath = sigObj.secureXMLVerifyFileToFile (signedXMLFile, outFilePath);
C Interface	ISignature_SecureXMLVerifyFileToFile (ISignature *pSig, BSTR signedXMLFile, BSTR outFilePath, BSTR * verificationResponseFilePath);
C++ Interface	pSig->SecureXMLVerifyToFile (BSTR signedXMLFile, BSTR outFilePath, BSTR * verificationResponseFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	verificationResponseFilePath = sigObj.SecureXMLVerifyFileToFile (signedXMLFile, outFilePath)
Supported Platforms	All

For detailed description please refer to SecureXMLVerify() method. SecureXMLVerifyFileToBuffer() method is a variation where the input is taken from a file instead of a memory buffer and the output is also written to a file. If outFilePath is NULL, the output is written to a temporary file, which is deleted during object destruction. Upon return verificationResponseFilePath is set to the complete path of the output file.

If invoked from a browser, the input parameter outFilePath is ignored and a temporary file is created instead.

SelectActiveCertificate

IDL File Declaration	[id(56), helpstring("method SelectActiveCertificate")] HRESULT SelectActiveCertificate([out, retval] BSTR *certID);
Java Interface	Not Supported
C Interface	ISignature_SelectActiveCertificate (ISignature *pSig, BSTR *certID);
C++ Interface	pSig-> SelectActiveCertificate (BSTR *certID);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	certID = sigObj. SelectActiveCertificate()
Supported Platforms	Windows Only

Presents the user with a certificate selection dialog box. The selected certificate is then becomes the active certificate and is used for all subsequent signing operations i.e. calls to all signature creation methods now perform their task silently. Upon return certID contains the certificate serial number for the seleted certificate. It is set to null if no certificate was selected.

SetActiveCertificate

IDL File Declaration	[id(28), helpstring("method SetActiveCertificate")] HRESULT SetActiveCertificate([in] BSTR certID, [out, retval] BOOL *status);
Java Interface	public int setActiveCertificate(String certID); int status = sigObj.setActiveCertificate(certID);
C Interface	ISignature_SetActiveCertificate (ISignature *pSig, BSTR certID, BOOL * status);
C++ Interface	pSig->SetActiveCertificate (BSTR certID, BOOL * status);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	status = sigObj.SetActiveCertificate(certID)
Supported Platforms	All

Sets the signer certificate to the certificate matching the given certificate serial number (certID) for all subsequent calls to all signature creation methods.

SetActivePEMFileCert

IDL File Declaration	[id(126), helpstring("method SetActivePEMFileCert")] HRESULT SetActivePEMFileCert([in] BSTR pemFileName, [in] BSTR pemPassword, [out,retval] BSTR* pemX509Cert);
Java Interface	public String setActivePEMFileCert(String pemFileName, String pemPassword); String x509Cert = sigObj.setActivePEMFileCert (pemFileName, pemPassword);
C Interface	ISignature_SetActivePFXFileCert (ISignature *pSig, BSTR pemFileName, BSTR pemPassword, BSTR *x509Cert);
C++ Interface	pSig->SetActivePFXFileCert (BSTR pemFileName, BSTR pemPassword, BSTR *x509Cert);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	x509Cert = sigObj.SetActivePFXFileCert (pemFileName, pemPassword);
Supported Platforms	Windows Only

SetActivePEMFileCert takes the PEM certificate file name and the password used during creating the PEM file as two input parameters and returns the corresponding Base64 encoded X509 certificate, which can be used to extract signer information. It sets this PEM key to be the active certificate for all subsequent calls to all signature creation functions. A call to SetActiveCertificate, SelectActiveCertificate or SetActivePFXFileCert must be made in order to change the active signer certificate. You may call SetActivePEMFileCert again to change the active certificate to another PEM file based certificate.

SetActivePFXB64Data

IDL File Declaration	[id(142), helpstring("method SetActivePFXB64Data")] HRESULT SetActivePFXB64Data([in] BSTR b64PfxData, [in] BSTR pfxPassword, [out,retval] BSTR* pfxX509Cert);
Java Interface	public String setActivePFXB64Data (String b64PfxData, String pfxPassword); String x509Cert = sigObj.setActivePFXB64Data (b64PfxData, pfxPassword);
C Interface	ISignature_SetActivePFXB64Data (ISignature *pSig, BSTR b64PfxData, BSTR pfxPassword, BSTR *x509Cert);
C++ Interface	pSig->SetActivePFXB64Data (BSTR b64PfxData, BSTR pfxPassword, BSTR *x509Cert);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	x509Cert = sigObj.SetActivePFXB64Data (b64PfxData, pfxPassword);
Supported Platforms	All

SetActivePFXB64Data takes base64 encoded PFX/P12 file data and the password used during creating the PFX file as two input parameters and returns the corresponding Base64 encoded X509 certificate, which can be used to extract signer information. It sets this PFX key to be the active certificate for all subsequent calls to all signature creation functions. A call to SetActiveCertificate, SelectActiveCertificate, SetActivePFXFileCert or SetActivePEMFileCert must be made in order to change the active signer certificate. You may call SetActivePFXB64Data again to change the active certificate to another base64 encoded PFX file data based certificate.

SetActivePFXFileCert

IDL File Declaration	[id(58), helpstring("method SetActivePFXFileCert")] HRESULT SetActivePFXFileCert([in] BSTR pfxFileName, [in] BSTR pfxPassword, [out, retval] BSTR *x509Cert);
Java Interface	public String setActivePFXFileCert(String pfxFileName, String pfxPassword); String x509Cert = sigObj.setActivePFXFileCert (pfxFileName, pfxPassword);
C Interface	ISignature_SetActivePFXFileCert (ISignature *pSig, BSTR pfxFileName, BSTR pfxPassword, BSTR *x509Cert);
C++ Interface	pSig->SetActivePFXFileCert (BSTR pfxFileName, BSTR pfxPassword, BSTR *x509Cert);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	x509Cert = sigObj.SetActivePFXFileCert (pfxFileName, pfxPassword);
Supported Platforms	All

SetActivePFXFileCert takes the PFX file name and the password used during creating the PFX file as two input parameters and returns the corresponding Base64 encoded X509 certificate, which can be used to extract signer information. It sets this PFX key to be the active certificate for all subsequent calls to all signature creation functions. A call to SetActiveCertificate, SelectActiveCertificate or SetActivePEMFileCert must be made in order to change the active signer certificate. You may call SetActivePFXFileCert again to change the active certificate to another PFX file based certificate.

SetStoreName

IDL File Declaration	[id(49), helpstring("method SetStoreName")] HRESULT SetStoreName([in] BSTR storeName);
Java Interface	public void setStoreName(String storeName); sigObj.setStoreName (storeName);
C Interface	ISignature_SetStoreName (ISignature *pSig, BSTR storeName);
C++ Interface	pSig->SetStoreName (BSTR storeName);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SetStoreName (storeName);
Supported Platforms	All

This method sets the current certificate store equal to storeName. The default value is the current user's "MY" certificate store. Invoking this method is almost close to reinitializing everything in the current object. All prior signature verification results etc. will be lost. In order to restore your certificate store to be your default store, invoke this method again with "MY" as the value for storeName.

Please invoke SetStoreName with "Netscape" as the store name parameter in order to use the Netscape certificate store. Please note that the NetscapeStorePassword property must to set to appropriate value if the Netscape certificate store is password protected before invoking SetStoreName("Netscape") method.

Sign

IDL File Declaration	[id(15), helpstring("method Sign")] HRESULT Sign([in] BSTR URI, [out, retval] BSTR *tempFileName);
Java Interface	public String sign(String URI); String tempFileName = sigObj.sign(URI);
C Interface	ISignature_Sign (ISignature *pSig, BSTR URI, BSTR *tempFileName);
C++ Interface	pSig->Sign (BSTR URI, BSTR *tempFileName);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	verificationResponse = sigObj.Sign (URI)
Supported Platforms	All

Signs the input URI and produces a detached signature or an enveloped signature or an enveloping signature depending on the setting of the EnvelopingFlag. The URI can either be a file name or a URL such as http://www.infomosaic.net. If the EnvelopingFlag is set to 2 i.e. an enveloped signature is requested, the URI must be a valid XML file (i.e. it can't be a URL).

The resulting signature is stored in a temporary file and the path to this file is returned in the return parameter tempFileName.

SignDataStr

IDL File Declaration	[id(87), helpstring("method SignDataStr")] HRESULT SignDataStr([in] BSTR dataStr, [out, retval] BSTR *signedXMLStr);
Java Interface	public String signDataStr(String dataStr); String signedXMLStr = sigObj.signDataStr (dataStr);
C Interface	ISignature_SignDataStr (ISignature *pSig, BSTR dataStr, BSTR *signedXMLStr);
C++ Interface	pSig->SignDataStr (BSTR dataStr, BSTR *signedXMLStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXMLStr = sigObj.SignDataStr (dataStr);
Supported Platforms	All

This mehod is used to sign arbitrary data string. The output XML Signature string contains the input data string as the content of <SignedObject> element. If the input string contains any characters not allowed in XML, they are XML encoded.

SignFile

IDL File Declaration	[id(47), helpstring("method SignFile")] HRESULT SignFile([in] BSTR inputFile, [in] BSTR outputFile);
Java Interface	public void signFile(String inputFile, String outputFile); sigObj.signFile (inputFile , outputFile);
C Interface	ISignature_SignFile (ISignature *pSig, BSTR inputFile, BSTR outputFile);
C++ Interface	pSig->SignFile (BSTR inputFile, BSTR outputFile);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.SignFile (inputFile, outputFile);
Supported Platforms	All

This method is useful for signing any arbitrary file data. It creates an enveloped signature and it can be used to create signed xml which can be signed by multiple people. Both inputFile and outputFile must be local files (can't be URL). The output of this method can be fed as an input to GetSignedFileObject function to retrieve the original file.

SignFiles

IDL File Declaration	[id(91), helpstring("method SignFiles")] HRESULT SignFiles([in] VARIANT fileList, [in] BSTR outFileName, [out, retval] BSTR *outFilePath);
Java Interface	public String signFiles(String [] fileList, String outFileName); outFilePath = sigObj.signFiles (fileList , outFileName);
C Interface	ISignature_SignFiles (ISignature *pSig, VARIANT fileList, BSTR outFileName, BSTR *outFilePath);
C++ Interface	pSig->SignFiles (VARIANT fileList, BSTR outFileName, BSTR *outFilePath);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	outFilePath = sigObj.SignFiles (fileList, outFileName);
Supported Platforms	All

This method is useful for signing any arbitrary file data. It creates an enveloped signature and it can be used to create signed xml which can be signed by multiple people. Both inputFile and outputFile must be local files (can't be URL). The output of this method can be fed as an input to GetSignedFileObject function to retrieve the original file.

SignHTML

IDL File Declaration	[id(43), helpstring("method SignHTML")] HRESULT SignHTML([in] IDispatch *document, [out, retval] BSTR *signedHTML);
Java Interface	public String signHTML(Object document); signedHTML = sigObj.signHTML (document);
C Interface	ISignature_SignHTML (ISignature *pSig, IDispatch *document, BSTR *signedHTML);
C++ Interface	pSig->SignHTML (IDispatch *document, BSTR *signedHTML);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedHTML = sigObj.SignHTML (document);
Supported Platforms	Windows Only

This method is useful for signing an HTML page along with any form data filled in by the user. The document input parameter must be an HTML document. The return string contains an XML with embedded object and the signature.

From a html page you can invoke this function as follows:

SigObj.SignHTML(document) or SigObj.SignHTML(window.document)

SignSignedInfoDigest

IDL File Declaration	[id(153), helpstring("method SignSignedInfoDigest")] HRESULT SignSignedInfoDigest([in] BSTR b64CertData, [in] BSTR b64SignedInfoDigest, [out,retval] BSTR* b64SigValXml);
Java Interface	public String signSignedInfoDigest (String b64CertData, String b64SignedInfoDigest); String signedXMLStr = sigObj.signSignedInfoDigest (b64CertData, b64SignedInfoDigest);
C Interface	ISignature_SignSignedInfoDigest (ISignature *pSig, BSTR b64CertData, BSTR b64SignedInfoDigest , BSTR *signedXMLStr);
C++ Interface	pSig->SignSignedInfoDigest (BSTR b64CertData, BSTR b64SignedInfoDigest, BSTR *signedXMLStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXMLStr = sigObj.SignSignedInfoDigest (b64CertData, b64SignedInfoDigest);
Supported Platforms	All

This mehod is part of the split signing mechanism supported by SecureXML. This method is typically invoked on the client side. The input to this method are the base64 encoded signer X509 certifcate data and the result to a call to GetSignedInfoDigest or GetSigned InfoDigestFromByteArray methods, which are typically invoked on server side. The output of this method is then passed to the server side code, which passes this information to ApplySignatureValue or ApplySignatureValueGetByteArray methods in order to complete the signature creation process. If b64CertData is null, a certificate selection dialog box is displayed for certificate selection.

SignXMLByteArray

IDL File Declaration	[id(129), helpstring("method SignXMLByteArray")] HRESULT SignXMLByteArray([in] VARIANT xmlByteArray, [in] BSTR signatureId, [out,retval] VARIANT* signedXmlByteArray);
Java Interface	public byte [] signXMLByteArray(byte [] xmlByteArray, String signatureId); byte [] signedXmlByteArray = sigObj.signXMLByteArray (xmlByteArray, signatureId);
C Interface	ISignature_SignXMLByteArray (ISignature *pSig, VARIANT xmlByteArray, BSTR signatureId, VARIANT* signedXmlByteArray);
C++ Interface	pSig->SignXMLByteArray (VARIANT xmlByteArray, BSTR signatureId, VARIANT* signedXmlByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXmlByteArray = sigObj.SignXMLByteArray (xmlByteArray, signatureId)
Supported Platforms	All

It expects a signature template as the input in *xmlByteArray*. It looks for a signature element with signatureId as the Id. If signatureId is blank, it looks for a signature element with no Id. The returned value contains the XML with signature values populated. The input template XML data must contain appropriate name space definition for the Signature element.

Here is an example template file (signature.tmpl):

```
<?xml version="1.0"?>
<Envelope xmlns="http://www.usps.gov/" xmlns:foo="http://www.usps.gov/foo" xml:lang="EN">
  <DearSir>foo</DearSir>
  <Body Id="SignonRq">bar</Body>
  <Notaries xmlns="" Id="notaries">
    <Notary name="Great, A. T."/>    <Notary name="Hun, A. T."/>
  </Notaries>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="MySignature">
  <SignedInfo>  <Reference URI="#notaries">  </Reference>  </SignedInfo>
  </Signature>
</Envelope>
```

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

SignXMLEnveloped

IDL File Declaration	[id(51), helpstring("method SignXMLEnveloped")] HRESULT SignXMLEnveloped([in] BSTR inputXML, [in] BSTR sigId, [out, retval] BSTR *signedXML);
Java Interface	public String signXMLEnveloped(String inputXML, String sigId); String signedXML = sigObj. signXMLEnveloped (inputXML, sigId);
C Interface	ISignature_SignXMLEnveloped (ISignature *pSig, BSTR inputXML, BSTR sigId, BSTR * signedXML);
C++ Interface	pSig->SignXMLEnveloped (BSTR inputXML, BSTR sigId, BSTR * signedXML);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXML = sigObj.SignXMLEnveloped (inputXML, sigId)
Supported Platforms	All

This is a special case of SignXMLStr where entire input XML is always signed using the enveloped signature transform. One can set SignatureID and Properties for this new signature being added. The function parses the input XML string but doesn't look for a SignatureID. It simply adds a fresh <Signature> element. This function is useful in Web Form signing situations where you have the XML as a string (not in a file, so you can't use Sign), all you want to do is sign the whole thing and you know that you don't need additional signatures to be added to this XML. The resulting signed XML can be directly sent to a database using appropriate SQL statements.

SignXMLEnvelopedByteArray

IDL File Declaration	[id(135), helpstring("method SignXMLEnvelopedByteArray")] HRESULT SignXMLEnvelopedByteArray([in] VARIANT inputXmlByteArray, [in] BSTR sigId, [out,retval] VARIANT* signedXmlByteArray);
Java Interface	public byte [] signXMLEnvelopedByteArray(byte [] inputXmlByteArray, String sigId); byte [] signedXmlByteArray = sigObj.signXMLEnvelopedByteArray (inputXmlByteArray, sigId);
C Interface	ISignature_SignXMLEnvelopedByteArray (ISignature *pSig, VARIANT inputXmlByteArray, BSTR sigId, VARIANT* signedXmlByteArray);
C++ Interface	pSig->SignXMLEnvelopedByteArray (VARIANT inputXmlByteArray, BSTR sigId, VARIANT* signedXmlByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXmlByteArray = sigObj.SignXMLEnvelopedByteArray (inputXmlByteArray, sigId)
Supported Platforms	All

This is a special case of SignXMLByteArray where entire input XML is always signed using the enveloped signature transform. One can set SignatureID and Properties for this new signature being added. The function parses the input XML byte array but doesn't look for a SignatureID. It simply adds a fresh <Signature> element. This function is useful in Web Form signing situations where you have the XML as a byte array (not in a file, so you can't use Sign), all you want to do is sign the whole thing and you know that you don't need additional signatures to be added to this XML. The resulting signed XML can be directly sent to a database using appropriate SQL statements.

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

SignXMLStr

IDL File Declaration	[id(40), helpstring("method SignXMLStr")] HRESULT SignXMLStr([in] BSTR xmlStr, [in] BSTR signatureId, [out, retval] BSTR *signedXMLStr);
Java Interface	public String signXMLStr(String xmlStr, String signatureId); String signedXMLStr = sigObj. signXMLStr (xmlStr, signatureId);
C Interface	ISignature_SignXMLStr (ISignature *pSig, BSTR xmlStr, BSTR signatureId, BSTR *signedXMLStr);
C++ Interface	pSig->SignXMLStr (BSTR xmlStr, BSTR signatureId, BSTR *signedXMLStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXMLStr = sigObj.SignXMLStr (xmlStr, signatureId)
Supported Platforms	All

It expects a signature template as the input in *xmlStr*. It looks for a signature element with signatureId as the Id. If signatureId is blank, it looks for a signature element with no Id. The returned value contains the XML with signature values populated.

The input template XML data must contain appropriate name space definition for the Signature element.

Here is an example template file (signature.tmpl):

```
<?xml version="1.0"?>
<Envelope xmlns="http://www.usps.gov/" xmlns:foo="http://www.usps.gov/foo" xml:lang="EN">
  <DearSir>foo</DearSir>
  <Body Id="SignonRq">bar</Body>
  <Notaries xmlns="" Id="notaries">
    <Notary name="Great, A. T."/>
    <Notary name="Hun, A. T."/>
  </Notaries>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="MySignature">
  <SignedInfo>
  <Reference URI="#notaries">
  </Reference>
  </SignedInfo>
  </Signature>
</Envelope>
```

SignXMLXpathStr

IDL File Declaration	[id(81), helpstring("method SignXMLXpathStr")] HRESULT SignXMLXpathStr([in] BSTR xmlStr,[in] BSTR xpathExp,[in] BSTR signatureId,[out, retval] BSTR *signedXMLStr);
Java Interface	public String signXMLXpathStr(String xmlStr, String xpathExp, String signatureId); String signedXMLStr = sigObj. signXMLXpathStr (xmlStr, xpathExp, signatureId);
C Interface	ISignature_SignXMLXpathStr (ISignature *pSig, BSTR xmlStr, BSTR xpathExp, BSTR signatureId, BSTR *signedXMLStr);
C++ Interface	pSig->SignXMLXpathStr (BSTR xmlStr, BSTR xpathExp, BSTR signatureId, BSTR *signedXMLStr);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXMLStr = sigObj.SignXMLXpathStr (xmlStr, xpathExp, signatureId)
Supported Platforms	All

This function is identical to SignXMLStr except that it allows for providing an XPATH expression which is used as an additional transform in addition to the enveloped signature transform. If a signature element with a matching signature Id is found, the XPATH expression is ignored and the signature is created as per the template provided in that signature element. Namespaces to be used for evaluating the XPATH expression can be provided by setting the XpathNamespace object property.

SignXMLXpathByteArray

IDL File Declaration	[id(134), helpstring("method SignXMLXpathByteArray")] HRESULT SignXMLXpathByteArray([in] VARIANT inputXmlByteArray, [in] BSTR xpathExp, [in] BSTR signatureId, [out,retval] VARIANT* signedXmlByteArray);
Java Interface	public byte [] signXMLXpathByteArray(byte [] inputXmlByteArray, String xpathExp, String signatureId); byte [] signedXmlByteArray = sigObj.signXMLXpathByteArray (inputXmlByteArray, xpathExp, signatureId);
C Interface	ISignature_SignXMLXpathByteArray (ISignature *pSig, VARIANT inputXmlByteArray, BSTR xpathExp, BSTR signatureId, VARIANT* signedXmlByteArray);
C++ Interface	pSig->SignXMLXpathByteArray (VARIANT inputXmlByteArray, BSTR xpathExp, BSTR signatureId, VARIANT* signedXmlByteArray);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	signedXmlByteArray = sigObj.SignXMLXpathByteArray (inputXmlByteArray, xpathExp, signatureId)
Supported Platforms	All

This function is identical to SignXMLByteArray except that it allows for providing an XPATH expression which is used as an additional transform in addition to the enveloped signature transform. If a signature element with a matching signature Id is found, the XPATH expression is ignored and the signature is created as per the template provided in that signature element. Namespaces to be used for evaluating the XPATH expression can be provided by setting the XpathNamespace object property.

A byte array is either a VARIANT with type (VT_ARRAY | VT_UI1) or a VARIANT with type (VT_VARIANT | VT_BYREF) where the VARIANT being referenced is of type (VT_ARRAY | VT_UI1).

ViewAnyCertificate

IDL File Declaration	[id(55), helpstring("method ViewAnyCertificate")] HRESULT ViewAnyCertificate([in] BSTR inputX509Data);
Java Interface	public void viewAnyCertificate(String inputX509Data) sigObj.viewAnyCertificate (inputX509Data);
C Interface	ISignature_ViewAnyCertificate (ISignature *pSig, BSTR inputX509Data);
C++ Interface	pSig->ViewAnyCertificate (BSTR inputX509Data);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ViewAnyCertificate (inputX509Data);
Supported Platforms	Windows Only

Pops a window with the certificate information. The input value is base64 encoded X509 data. This certificate need not be installed in the local machine.

ViewCertificate

IDL File Declaration	[id(25), helpstring("method ViewCertificate")] HRESULT ViewCertificate([in] BSTR certID);
Java Interface	public void viewCertificate(String certID) sigObj.viewCertificate (certID);
C Interface	ISignature_ViewCertificate (ISignature *pSig, BSTR certID);
C++ Interface	pSig->ViewCertificate (BSTR certID);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigObj.ViewCertificate (certID);
Supported Platforms	Windows Only

Pops a window with the certificate information. The input value is base64 encoded certificate serial number of a certificate installed in the local machine store.

Verify

IDL File Declaration	[id(18), helpstring("method Verify")] HRESULT Verify([in] BSTR signatureFileName, [out, retval] BOOL *sigStatus);
Java Interface	public int verify(String signatureFileName); int sigStatus = sigObj.verify (signatureFileName);
C Interface	ISignature_Verify (ISignature *pSig, BSTR signatureFileName, BOOL *sigStatus);
C++ Interface	pSig->Verify (BSTR signatureFileName, BOOL *sigStatus);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigStatus = sigObj.Verify (signatureFileName)
Supported Platforms	All

It verifies all signatures present in the input signature file given as signatureFileName. It also sets various signature properties which provide additional information about the signature. The return value is set to 1 if the signature verification was successful and it is 0 otherwise.

VerifyActiveCertificate

IDL File Declaration	[id(70), helpstring("method VerifyActiveCertificate")] HRESULT VerifyActiveCertificate([out, retval] BOOL *result);
Java Interface	public int verifyActiveCertificate(); int pVal = sigObj.verifyActiveCertificate ();
C Interface	ISignature_VerifyActiveCertificate (ISignature *pSig, BOOL *pVal);
C++ Interface	pSig->VerifyActiveCertificate (BOOL *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.VerifyActiveCertificate ()
Supported Platforms	All

This method is checks the validity of the currently active certificate. At least one of the various certificate validation flags must be set to 1 in order for this method to provide any meaningful result. The result is 1 if the certificate is valid, it is 0 otherwise.

VerifyDetached

IDL File Declaration	[id(22), helpstring("method VerifyDetached")] HRESULT VerifyDetached([in] BSTR signatureFileName, [out, retval] BOOL *sigStatus);
Java Interface	public int verifyDetached(String signatureFileName); int sigStatus = sigObj.verifyDetached (signatureFileName);
C Interface	ISignature_VerifyDetached (ISignature *pSig, BSTR signatureFileName, BOOL *sigStatus);
C++ Interface	pSig->VerifyDetached (BSTR signatureFileName, BOOL *sigStatus);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigStatus = sigObj.VerifyDetached (signatureFileName)
Supported Platforms	All

It should be called after a call to Verify fails with DEREF_URI_ERROR.

VerifyPFXCertCRL

IDL File Declaration	[id(67), helpstring("method VerifyPFXCertCRL")] HRESULT VerifyPFXCertCRL([in] BSTR pfxFileName, [in] BSTR pfxPassword, [in] BSTR atTime, [in] long timeFormat, [out, retval] BOOL *pVal);
Java Interface	public int verifyPFXCertCRL(String pfxFileName, String pfxPassword, String atTime, int timeFormat); int pVal = sigObj.verifyPFXCertCRL(pfxFileName, pfxPassword, atTime, timeFormat);
C Interface	ISignature_VerifyPFXCertCRL (ISignature *pSig, BSTR pfxFileName, BSTR pfxPassword, BSTR atTime, long timeFormat, BOOL *pVal);
C++ Interface	pSig->VerifyPFXCertCRL (BSTR pfxFileName, BSTR pfxPassword, BSTR atTime, long timeFormat, BOOL *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.VerifyPFXCertCRL (pfxFileName, pfxPassword, atTime, timeFormat)
Supported Platforms	All

This method is checks the certificate contained in the pfxFileName against its CRL for revocation at time equal to atTime. If the certificate was revoked then pVal is set to FALSE (=0). It the certificate was valid then pVal is set to TRUE (=1). If the parameter atTime is NULL, the current system time is used.

The following time formats are supported:

Time Format Value	**Format**
TIME_RFC or 0	This is the format returned by NIST time server at http://time-b.timefreq.bldrdoc.gov:13 also know as RFC-867 format. JJJJJ YR-MO-DA HH:MM:SS TT L H msADV UTC(NIST) OTM Example: 52478 02-07-23 22:37:41 50 0 0 558.1 UTC(NIST) where: **JJJJJ** is the Modified Julian Date (MJD). The MJD is the last five digits of the Julian Date, which is simply a count of the number of days since January 1, 4713 B.C. To get the Julian Date, add 2.4 million to the MJD. (This number is ignored but must be present). **YR-MO-DA** is the date. It shows the last two digits of the year, the month, and the current day of month. **HH:MM:SS** is the time in hours, minutes, and seconds. The time is always sent as Coordinated Universal Time (UTC). An offset needs to be applied to UTC to obtain local time. For example, Mountain Time in the U. S. is 7 hours behind UTC during Standard Time, and 6 hours behind UTC during Daylight Saving Time. **TT** is a two digit code (00 to 99) that indicates whether the

United States is on Standard Time (ST) or Daylight Saving Time (DST). It also indicates when ST or DST is approaching. This code is set to 00 when ST is in effect, or to 50 when DST is in effect. During the month in which the time change actually occurs, this number will decrement every day until the change occurs. For example, during the month of October, the U.S. changes from DST to ST. On October 1, the number will change from 50 to the actual number of days until the time change. It will decrement by 1 every day until the change occurs at 2 a.m. local time when the value is 1. Likewise, the spring change is at 2 a.m. local time when the value reaches 51.

L is a one-digit code that indicates whether a leap second will be added or subtracted at midnight on the last day of the current month. If the code is 0, no leap second will occur this month. If the code is 1, a positive leap second will be added at the end of the month. This means that the last minute of the month will contain 61 seconds instead of 60. If the code is 2, a second will be deleted on the last day of the month. Leap seconds occur at a rate of about one per year. They are used to correct for irregularity in the earth's rotation. The correction is made just before midnight UTC (not local time).

H is a health digit that indicates the health of the server. If H=0, the server is healthy. If H=1, then the server is operating properly but its time may be in error by up to 5 seconds. This state should change to fully healthy within 10 minutes. If H=2, then the server is operating properly but its time is known to be wrong by more than 5 seconds. If H=4, then a hardware or software failure has occurred and the amount of the time error is unknown.

msADV displays the number of milliseconds that NIST advances the time code to partially compensate for network delays. The advance is currently set to 50.0 milliseconds.
The label UTC(NIST) is contained in every time code. It indicates that you are receiving Coordinated Universal Time (UTC) from the National Institute of Standards and Technology (NIST).
OTM (on-time marker) is an asterisk (*). The time values sent by the time code refer to the arrival time of the OTM. In other words, if the time code says it is 12:45:45, this means it is 12:45:45 when the OTM arrives.

TIME_VB_NOW or 1

07/24/2002 3:44:13 PM
As returned by VB Script **Now** function

TIME_JS_UTC or 2

Wed, 24 Jul 2002 22:44:12 UTC

As returned by JavaScript toUTCString method on Date object.

```
var SigDate;
SigDate = new Date();
SigDate.toUTCString() == atTime
```

VerifyX509CertCRL

IDL File Declaration	[id(66), helpstring("method VerifyX509CertCRL")] HRESULT VerifyX509CertCRL([in] BSTR certData, [in] BSTR atTime, [in] long timeFormat, [out, retval] BOOL *pVal);
Java Interface	public int verifyX509CertCRL(String certData, String atTime, int timeFormat); int pVal = sigObj.verifyX509CertCRL (certData, atTime, timeFormat);
C Interface	ISignature_VerifyX509CertCRL (ISignature *pSig, BSTR certData, BSTR atTime, long timeFormat, BOOL *pVal);
C++ Interface	pSig->VerifyX509CertCRL(BSTR certData, BSTR atTime, long timeFormat, BOOL *pVal);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	pVal = sigObj.VerifyX509CertCRL (certData, atTime, timeFormat)
Supported Platforms	All

This method is checks the base64 encoded X509 certificate contained in certData against its CRL for revocation at time equal to atTime. If the certificate was revoked then pVal is set to FALSE (=0). It the certificate was valid then pVal is set to TRUE (=1). If the parameter atTime is NULL, the current system time is used.

For supported time formats, please see VerifyPFXCertCRL above.

VerifyXMLStr

IDL File Declaration	[id(57), helpstring("method VerifyXMLStr")] HRESULT VerifyXMLStr([in] BSTR signedXML, [out, retval] BOOL *sigStatus);
Java Interface	public int verifyXMLStr(String signedXML); int sigStatus = sigObj.verifyXMLStr (signedXML);
C Interface	ISignature_VerifyXMLStr (ISignature *pSig, BSTR signedXML, BOOL *sigStatus);
C++ Interface	pSig->VerifyXMLStr (BSTR signedXML, BOOL *sigStatus);
VB/VBScript/JavaScript/ C#.NET/VB.NET Interface	sigStatus = sigObj.VerifyXMLStr (signedXML)
Supported Platforms	All

This method is identical to Verify in all aspects except that it takes an XML string as the input parameter and not a file name.

SecureXML Java Applet Application Programming Interface (API) Reference

Overview

SecureXML Java Applet encapsulates the SecureXML Signature object, SecureXML SmartCard object and Netscape JavaScript classes for communicating with the browser using JavaScript call back methods.

The main entry class for the SecureXML Java Applet is ***infomosaic.securexml.applet.XMLSignApplet***. The following HTML element is needed to make browsers (both IE and Firefox) to download and initialize SecureXML Java Applet:

```
<applet codebase = "."
                  archive  = "securexmlapplet.jar"
                  code     = "infomosaic.securexml.applet.XMLSignApplet"
                  id="securexmlApplet"
                  name="securexmlApplet"
                  scriptable="true"
                  width    = "0"
                  height   = "0"
                  hspace   = "0"
                  vspace   = "0"
                  align    = "middle"
                  MAYSCRIPT VIEWASTEXT>
            <PARAM name="cabbase" value="securexmlapplet.cab" />
            </applet>
```

The above <applet> tag assumes that the files securexmlapplet.jar and securexmlapplet.cab are present in the same directory as the page on which the tag appears. If that is not the case, you must modify the codebase attribute to show the path relative to the current document. The `<PARAM name="cabbase" value="securexmlapplet.cab" />` portion is used only by Microsoft JVM as it loads a signed cab file instead of a signed jar for the applet.

The SecureXML Java Applet calls a JavaScript function “enableLinks()” immediately after its initialization has completed. Hence the page invoking SecureXML Java Applet must define a JavaScript function called “enableLinks()” and try using the applet object only after enableLinks() has been called by the applet. If the HTML page has user controls such as buttons which lead to applet method invocation, you can have those buttons disabled on page load and then enable them from within the “enableLinks()” method. The following HTML code illustrates this usage:

```
<html>
      <body onLoad="disableLinks()">

<applet codebase = "."
                  archive  = "securexmlapplet.jar"
                  code     = "infomosaic.securexml.applet.XMLSignApplet"
                  id="securexmlApplet"
                  name="securexmlApplet"
                  scriptable="true"
                  width    = "0"
                  height   = "0"
                  hspace   = "0"
                  vspace   = "0"
                  align    = "middle"
                  MAYSCRIPT VIEWASTEXT>
            <PARAM name="cabbase" value="securexmlapplet.cab" />
```

```
            </applet>

<Script language="JavaScript">
var newLink;
var linksEnabled=false;
var sig;
function disableLinks(){
   linksEnabled=false;
   var aLnk=document.getElementsByName("buttons");
   for (i=0; i<aLnk.length; i++) {
     aLnk[i].disabled = true;
   }
}

function enableLinks(){
   linksEnabled=true;
   var aLnk=document.getElementsByName("buttons");
   for (i=0; i<aLnk.length; i++) {
     aLnk[i].disabled = false;
   }
}
function invokeSign()
{
      //See code below HTML section
}
</Script>
            <form name="passwordForm">
            <b>Please enter Cac password:</b>
            <input type=text name="cacPassword" maxlength=50 id="cacPassword"
size="50">
            <input type="button" name="buttons" value="Click Me"
onclick="return invokeSign()">
            </form>
            <div id="CacOutputArea"></div>
            <div id="signatureOutputArea"></div>
      </body>
</html>
```

The SecureXML Java Applet provides two methods which allow access to the SecureXML Signature and SecureXML Smartcard objects.

```
sig = securexmlApplet.getSignature(); // Gets an instance of the SecureXML
Signature object
cac = securexmlApplet.getSmartcard(); // Gets an instance of the SecureXML
Smartcard object
```

Once you have reference to the SecureXML Signature object you can now invoke all the Java APIs provided for it from JavaScript. Please note that the API syntax for a SecureXML Signature object obtained from SecureXML Java Applet is that of Java Language as described in the SecureXML Digital Signature API section of this guide and not the one for JavaScript (which requires SecureXML Digital Signature ActiveX to be installed on the client computer). The following example shows this difference:

```
<Script language="JavaScript">

var sig1;
var sig2;
```

```
sig1 = new ActiveXObject(“XMLSign.Signature”);
sig2 = securexmlApplet.getSignature();

// Getting certificate count

certCount1 = sig1.CertificateCount; // sig1 is SecureXML ActiveX object
certCount2 = sig2.getCertificateCount(); // sig2 is SecureXML Java object
</Script>
```

The onclick event handler for the button in the above HTML can be coded as follows:

```
function invokeSign()
{
      var certCount=0;
      var certId = new String();
        var ainfo = document.securexmlApplet.getAppletInfo();
        sig = document.securexmlApplet.getSignature();
        cac = document.securexmlApplet.getSmartcard();
      try
      {
            cac.connectToCard("Some Card"); // “Some Card” input parameter is
ignored.
            // logonToCard is optional step. If the user has already
authenticated to the CAC then it is not required.
            cac.logonToCard(passwordForm.cacPassword.value);
            CacOutputArea.innerText = getAllInfoFromCac(cac);

            sig.selectActiveCertificate();
            signatureOutputArea.innerText = sig.signDataStr("This is a test")
+ sig.getError();
      }
      catch(Exception)
      {
            alert(ainfo + ": Caught exception " + Exception.toString());
            return false;
      }
      return true;
}
```

SecureXML Smartcard Object Methods

The following is a list of methods exposed by the SecureXML Smartcard object that can be used to access the data in the CAC.

- getFirstName()
  Retrieves the first name of the cardholder.
- getMiddleName()
  Retrieves the middle name of the cardholder.
- getLastName()
  Retrieves the last name of the cardholder.
- getCadency()
  Retrieves the suffix name of the cardholder.
- getPersonIdentifier()
  Retrieves the person identifier (e.g. SSN) of the cardholder.
- getDateOfBirth()
  Retrieves the date of birth of the cardholder.
- getGenderCode()
  Retrieves the gender of the cardholder.
- getPersonIdentifierTypeCode()
  Retrieves the type of person identifier of the cardholder.
- getBloodTypeCode()
  Retrieves the blood type code for the cardholder.
- getPersonDoDEDI()
  Retrieves the Department of Defense Electronic Data Interchange (EDI) number for the cardholder.
- getOrganDonorCode()
  Retrieves the organ donor code for the cardholder.
- getCardIssueDate()
  Retrieves the date the card was issued.
- getCardExpiryDate()
  Retrieves the expiration date of the card.
- getDateCacDataLoaded()
  Retrieves the date the data was loaded on the CAC.
- getDateCacDataExpires()
  Retrieves the date the data on the CAC expires.
- getExchangeCode()
  Retrieves the exchange code for the cardholder.
- getCommissaryCode()
  Retrieves the commissary code for the cardholder.
- getMWRCode()
  Retrieves the Morale, Welfare and Recreation (MWR) code for the cardholder.
- getNonMedicalBenefitsEndDate()
  Retrieves the non-medical benefits end date for the cardholder.

- getDirectCareEndDate()
  Retrieves direct care end date for the cardholder.
- getCHCCode()
  Retrieves the civilian health care code for the cardholder.
- getDirectCareCode()
  Retrieves the direct care code for the cardholder.
- getCHCEndDate()
  Retrieves the civilian health care end date for the cardholder.
- getMealPlanCode()
  Retrieves the meal plan code for the cardholder.
- getDoDContractorFunctionCode()
  Retrieves the DoD Contractor Function Code for the cardholder.
- getUSGovernmentAgencyCode()
  Retrieves the U. S. Government agency code for the cardholder.
- getBranchOfServiceCode()
  Retrieves the branch of service for the cardholder.
- getPayGrade()
  Retrieves the pay grade code for the cardholder.
- getRank()
  Retrieves the rank code for the cardholder.
- getPersonnelCategoryCode()
  Retrieves the personnel category code for the cardholder.
- getNonUSGovernmentAgencyCode()
  Retrieves the non-U. S. Government agency code for the cardholder.
- getPayPlan()
  Retrieves the pay plan code for the cardholder.
- getPersonnelEntitlementConditionTypeCode()
  Retrieves the personnel entitlement condition type code for the cardholder.
- getCN(int locationIndex)
  Retrives the CN from the certificate stored on the CAC at location corresponding to locationIndex, which can take values 0, 1 or 2 for locations 0100, 0101 or 0102 respectively.
- getEmailAddress(int locationIndex)
  Retrives the enail address from the certificate stored on the CAC at location corresponding to locationIndex, which can take values 0, 1 or 2 for locations 0100, 0101 or 0102 respectively.
- connectToCard(String readerName)
  Connects to the card, using the readerName. The parameter readerName is currently ignored and the connection is made to the first available CAC reader.
- logonToCard(String password)
  Logs on to the card. Calling this method is not required if the user has already authecticated to the CAC.

Appendix A: Error Codes

```
CERT_CHAIN_ERROR            1    // Certificate chain is not valid
BASE64_ERROR                2    // Base64 encoding or decoding error
C14_ERROR                   3    // Canonicalization error
PRIVATEKEY_ERROR            4    // Cannot acquire private key
XMLSIGLIB_ERROR             5    // XML Signature library error
INVALID_SIGMETHOD           6    // Cannot detect signature method
MEMORY_FAULT                7    // Memory fault (malloc)
INVALID_SIG_ALG             8    // Unsupported signature algorithm
INVALID_PROVIDER            9    // Unsupported provider
INVALID_TRANSFORM           10   // Unsupported transformation
INVALID_DIGEST_METHOD       11   // Wrong algorithm for digest method
IO_ERROR                    12   // IO error
NO_CERTIFICATE              13   // no certificate provided
INVALID_XML_SIGNATURE       14   // Invalid XML structure of <Signature>
SIGNATURE_NTE_ERROR         15   // Signature validation failed: NTE errors. This is a Windows error
XML_PARSER_ERROR            16   // XML parser error
TRANSFORM_ERROR             17   // transformation error
I2O_ERROR                   18   // Integer to Big-endian convertion failed
NO_CERT_STORE               19   // No specified store found for the current user
CAPI_CREATE_HASH            20   // CryptAPI error: Create Hash Failed
DEREF_URI_ERROR             21   // Dereference for URI failed
DIGEST_ERROR                22   // digest validation or calculation failed
KEY_IMPORT_ERROR            23   // Key import failed
SIGN_FAILED                 24   // Sign procedure failed
CERT_GET_PK_ERROR           25   // Get PK from certificate failed
NO_XML_OBJECT               26   // No object #OBJECT in XML
FILE_NOT_FOUND              27   // File not found
CERT_NOT_FOUND              28   // Certificate not found by ID
NEWKEYSET_ERROR             29   // CryptAcquireContext with new keyset failed
CERT_GET_DETAILS_ERROR      30   // Get issuer/subject from certificate failed
PUBLIC_KEY_NOT_MATCH        31   // public key from certificate is not the same as in XML
INVALID_XML                 32   // given XML file is invalid
XML_ERROR                   33   // XML/Xpath processor failed
PK_SIZE_TOO_BIG             34   // Must increase MAX_MODULUS_SIZE in cr.h
DECODE_ERROR                35   // BASE64 little-endian -> BYTE *big-endian decode failed
XPOINTER_NOT_IMPLEMENTED    36   // Xpointer not implemented
XPATH_ERROR                 37   // XPath error
RELATIVE_NS_FOUND           38   // c14n error: found relative namespace
XML_DOC_INIT_ERROR          39   // must free pXmlDoc before call xsCoreInitXmlDocument()
HTTP_ERROR                  40   // URI cannot be fetched
GENERATE_KEY_ERROR          41   // Generate RC2 key for HMAC failed
HMAC_KEY_SIZE_TOO_SMALL     42   // Generate RC2 key for HMAC failed because of key length
                                 // (password is too big for default key)
NO_KEY                      43   // Unable to verify - no key or certificate found
NO_VERIFICATION_LIC         44   // Signature verification license for SecureXML not found
NO_SIGNING_LIC              45   // Signature creation license for SecureXML not found
NO_SECUREXML_LIC            46   // License for SecureXML not found
INVALID_SECUREXML_LIC       47   // License for SecureXML is not valid
DLL_NOT_REGISTERED          48   // SecureXML dll not registered
NO_SIGNATURE_DATA           49   // There is no signature verification data available
SIG_INDEX_ERROR             50   // Signature Index >= SignatureCount
PROP_INDEX_ERROR            51   // Property Index >= PropertyCount for the given Signature Index
URI_INDEX_ERROR             52   // URI Index >= UriCount or FailedUriCount for the given signature index
CERT_INDEX_ERROR            53   // Certificate Index >= Certificate Count
CERT_VAL_INDEX_ERROR        54   // Certificate Value Index can only be 1,2,3 or 4
DOC_PATH_NOT_FOUND          55   // No SignedDocumentPath info exists for the given sig/uri index
SIG_NOT_FOUND               56   // No signature with the matching signature Id was found
PFX_BAD_PASSWORD            57   // Invalid password provided for PFX file
```

```
PFX_IMPORT_FAILED                  58    // Import failed for PFX certificate
PFX_INVALID_CERTIFICATE            59    // No private key found found in PFX certificate or other validation error
PFX_EXPORT_FAILED                  60    // Export failed for PFX certificate
CERT_TRUST_ERROR                   61    // Certificate trust/CRL check failed
FILE_PATH_TOO_LONG                 62    // The file path is >= MAX_PATH
FILE_PATH_NOT_SET                  63    // The file path hasn't been set yet
LICENSE_EXPIRED                    64    // The license has expired
INVALID_CSP                        65    // The given CSP value was invalid
NO_SECUREXML_CONFIG                66    // Config file for SecureXML not found
NO_HMAC_PASSWORD                   67    // No Hmac password was provided & no physical signature used
NULL_DATA_STRING                   68    // The data string provided is of zero length
DUPLICATE_SIG_ID                   69    // There is already a signature with matching signature Id present
INVALID_SDATA_NS                   70    // There is no XML element SignedObject in the infomosaic name space
INVALID_VARIANT                    71    // The Variant contains unsupported data type
CRL_BAD_SIGNATURE                  72    // CRL has invalid signature
CRL_ERROR                          73    // generic error on CRL validation
CRL_NOT_FOUND                      74    // no CRL related data found for the certificate/chain
CRL_CERT_REVOKED                   75    // certificate revoked according to CRL
CERT_TIME_INVALID                  76    // At least one of the certificates in the chain has invalid time
CERT_TIME_NOT_NESTED               77    // The time nesting of the certificate chain is invalid
CERT_SIGNATURE_INVALID             78    // At least one of the certificates in the chain has invalid signature
CERT_INVALID_USAGE                 79
CERT_PARTIAL_CHAIN                 80
CERT_UNTRUSTED_ROOT                81
CERT_TRUST_CYCLIC                  82

CERT_INVALID_CHAIN                 83    // Issuer name <> Subject name of issuer certificate
BASIC_CONSTRAINT_ERR               84    // Either the intermediate certificate does not
                                         // have a basicConstraints extension or it is
                                         // not marked critical or the cA is set to false

CERT_KEY_USAGE_ERR                 85    // If the application encounters an intermediate certificate in the
                                         // certificate path that has the key usage extension present with the
                                         // keyCertSign bit set to true and the basic constraints extension
                                         // present, the application must ensure that the certificate has the cA
                                         // component of the basic constraints extension set to TRUE.
CERT_POLICY_ERROR                  86    // Certificate policy validation failed
CRL_TOO_OLD                        87    // The nextUpdate time is earlier than current time
INVALID_ENC_ALG                    88    // Unsupported encryption algorithm
DECRYPTION_FAILED                  89    // Probably wrong certificate used for decryption
CERT_PATHLEN_ERR                   90    // There is a violation of the path length constraint in the certificate chain
NO_ENCRYPT_LIC                     91    // The license to create encrypted documents not present
NO_DECRYPT_LIC                     92    // The license to decrypt encrypted documents not present
CAM_CERT_REVOKED                   93    // Certificate is revoked as reported by the CAM server
CAM_CERT_EXPIRED                   94    // Certificate is expired as reported by the CAM server
CAM_CERT_SUSPENDED                 95    // Certificate is suspended as reported by the CAM server
CAM_CERT_FAILED_VERIFICATION 96          // Certificate failed verification for some reason as reported by the CAM server
CAM_CERT_ISSUER_NOT_FOUND 97             // Certificate issuer not found as reported by the CAM server
CAM_CERT_NOT_PARSABLE              98    // Certificate is not parsable as reported by the CAM server
CAM_CA_SYSTEM_BUSY                 99    // CA is too busy to respond as reported by the CAM server
CAM_CA_TIMEOUT                     100   // CA timed out during certification validation as reported by the CAM server
CAM_CA_CERT_UNKNOWN                101   // CA claims they did not issue this cert as reported by the CAM server
CAM_CA_REQ_BAD                     102   // comm or protocol problem between CAM and CA as reported by the CAM
                                         // server
CAM_CA_BAD_RESP_SIG                103   // invalid signature on CA response as reported by the CAM server
CAM_INTERNAL_ERROR                 104   // CAM internal error
CAPI_SET_HASH_PARAM                105           // CryptAPI error: Set Hash Param Failed
CAPI_HASH_DATA                     106           // CryptAPI error: CryptHashData Failed
CAPI_GET_HASH_PARAM                107           // CryptAPI error: CryptGetHashParam Failed
CAPI_CREATE_CERT_CNTX              108           // CryptAPI error: CertCreateCertificateContext Failed
CAPI_GET_USER_KEY                  109           // CryptAPI error: CryptGetUserKey Failed
CAPI_EXPORT_KEY                    110           // CryptAPI error: CryptExportKey Failed
```

CAPI_SIGN_HASH	111	// CryptAPI error: CryptSignHash Failed
CAPI_OPEN_STORE	112	// CryptAPI error: CertOpenStore Failed
CAPI_ADD_CERT	113	// CryptAPI error: CertAddEncodedCertificateToStore Failed
CAPI_GET_CERT_CHAIN	114	// CryptAPI error: CertGetCertificateChain Failed
CAPI_DECODE_OBJ	115	// CryptAPI error: CryptDecodeObject Failed
CAPI_GET_CERT_CNTX_PROP	116	// CryptAPI error: CertGetCertificateContextProperty Failed
CAPI_ADD_CERT_CNTX	117	// CryptAPI error: CertAddCertificateContextToStore Failed
CRSIGN_FAILED	118	// crSign Failed, Please inform Infomosaic Support
CAPI_ACQUIRE_CNTX	119	// CryptAcquireContext Failed
CAPI_DECRYPT_MSG	120	// CryptDecryptMessage Failed
CAPI_ENCRYPT_MSG	121	// CryptEncryptMessage Failed
ADO_ERROR	122	// Problem accessing CRL Cache database
OCSP_ERROR	123	// Generic OCSP Error (probably OCSP Server is not reachable)
OCSP_CERT_UNKNOWN	124	// Certificate issuer is not known to the OCSP responder
OCSP_CERT_REVOKED	125	// Certificate is revoked as per OCSP response
OCSP_CA_UNTRUSTED	126	// The OCSP Response is not signed by a trusted authority
NSS_NOT_SUPPORTED	127	// This call is currently not supported for NSS API
TIME_STAMP_FAILED	128	// Could not obtain time stamp from the time stamp server

Appendix B

Deploying SecureXML in Server & Client Configurations and Explanations for Various DLLs & Jars

The following are the various files included with SecureXML SDK which are relevant for deploying SecureXML in a runtime environment:

- XMLSign.dll (Windows only)
- SigWinImage.dll (Windows only)
- SignatureL.dll (Windows only)
- libSignatureL.so (Linux only)
- libxmlsig.so (Linux only)
- libSignatureL.jnilib (Mac OS X only)
- libxmlsig.dylib (Mac OS X only)
- Securepad.dll (Windows only)
- ocspAX.dll (Windows only)
- NetscapeCert.exe (Windows only)
- PemUtil.dll (Windows only)
- gdiplus.dll (Windows only)
- securexml.jar (All platforms)
- securexmlapplet.jar (Windows only)
- securexmlapplet.cab (Windows only)
- SecureXML.mdb (Windows only)
- ErrorCodes.mdb (Windows only)

XMLSign.dll is the main Windows file which must be included in both server and client side deployments on Windows. SignatureL.dll is used by securexml.jar on Windows for supporting SecureXML Java APIs. If your application is written in Java and it does not use any other scripting or compiled languages to access SecureXML APIs, you need to include securexml.jar along with SignatureL.dll (XMLSign.dll is not needed). On Linux platforms you need libSignatureL.so and libxmlsig.so and on Mac OS X libSignatureL.jnilib along with libxmlsig.dylib is needed.

The SecureXML Java Applet is provided in two different packagings, securexmlapplet.jar for use with SUN JVM and securexmlapplet.cab for use with Microsoft JVM.

Gdiplus.dll is used for all signature/window image related functionality offered by SecureXML on Windows platform (not supported on non-Windows platforms). If your application does not capture signature/window images, you do not need to include this file. Gdiplus.dll is a windows component and is typically already present on most systems and is not required to be included with your deployment. It is included with SecureXML SDK since some Windows 98/2K version machines may not have it in their Windows\Systems32 directory. If your application needs to support Windows 98/2K, it is a good idea to include this file as part of your deployment. SigWinImage.dll will not register unless gdiplus.dll is found in either the same directory as SigWinImage.dll or in one of the directories pointed to by the PATH environment variable.

Securepad.dll is the component which is responsible for interacting with your Wintab32 compatible signature image capture device. It requires Wintab32.dll (provided by your signature pad manufacturer) before it can be registered. If you install the signature pad software after deploying SecureXML, you would need to manually register securepad.dll by issueing a “regsrv32 securepad.dll” command from a command prompt. Include securepad.dll for client side deployment if a signature capture device needs to be supported. For mouse and file based signature image capture SigWinImage.dll is sufficient.

If OCSP needs to be used for certificate validation, ocspAX.dll must be included and it needs to be registered. If CAM server is used, the responsibility to make the OCSP request is shifted to the CAM server and hence ocspAX.dll is not needed. On Unix platforms, OCSP support requires OpenSSL to be present.

If you need to support Netscape certificate stores in your application, you would need to include NetscapeCert.exe with your client side deployments. Please note NetscapeCert.exe needs to be registered before it can be used with SecureXML. NetscapeCert.exe depends on various dlls included with Netscape version 7.1 or later. Please make sure that the PATH environment variable contains the Netscape installation directory (C:\Program Files\Netscape\Netscape, by default) so that the

Windows runtime can find the various dlls needed. If the user machine does not have Netscape 7.1 or later, NetscapeCert.exe registration will fail.

Include PemUtil.dll with your shipment if you need to support PEM formatted certificates.

If your application is written in Java, you would need to include securexml.jar and SignatureL.dll for the server side and securexmlapplet.jar and securexmlapplet.cab for the client side with your deployment.

If you would like to use the CRL cache feature of SecureXML, you need to either ship SecureXML.mdb or use another database with the same tables as SecureXML.mdb file. The file ErrorCodes.mdb is provided for your convenience and is not used by SecureXML runtime. Your application may refer to it for providing details of errorcodes reported by SecureXML.

Server vs. Client Side Deployment

The key difference between the server side and client side deployment is with respect to the usage of GUI for certificate selection and signature capture and hence Securepad.dll and NetscapeCert.exe files not relevant to server side deployment.

Appendix C: Additional Information Related to DoD PKI Settings and Compliance

Overview

This section describes how to install, configure, and use SecureXML within the DOD PKI.

This section covers:

- Installing DOD PKI trust points and removing non-DOD PKI trust points
- Importing existing keys and certificates
- Installing Uniform Resource Indicators for DOD PKI services, such as obtaining certificates for other entities and performing status checking
- Configuring SecureXML properly to be interoperable with the DOD PKI according to DOD requirements

Installing DOD PKI trust points

What is a Trust Point?

A trust point is a root certificate of a certification authority. If you examine a certificate using Internet Explorer, and you view the certificate chain, the trust point is the last certificate in the certificate chain starting from the end user certificate.

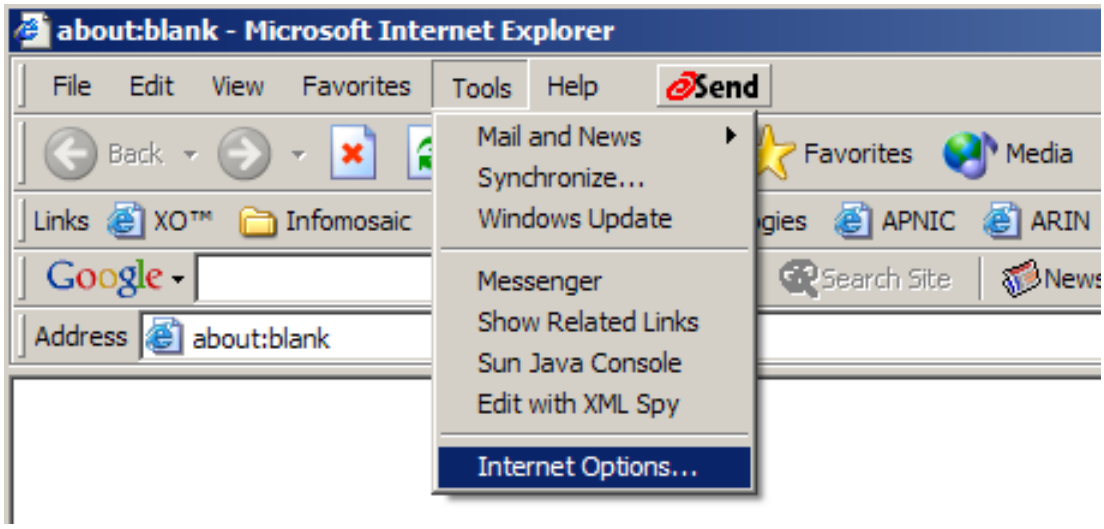
How does an application specify trust points in order to enforce DoD Trust Point compliance?

Your application would need to set DoDCompliance to 1 and set TrustedRoots to appropriate file path containing the DoD trusted root certificates. The certificates could either be base64 encoded or DER encoded. Please refer to documentation for DoDCompliance and TrustedRoots properties for code examples of using these properties.

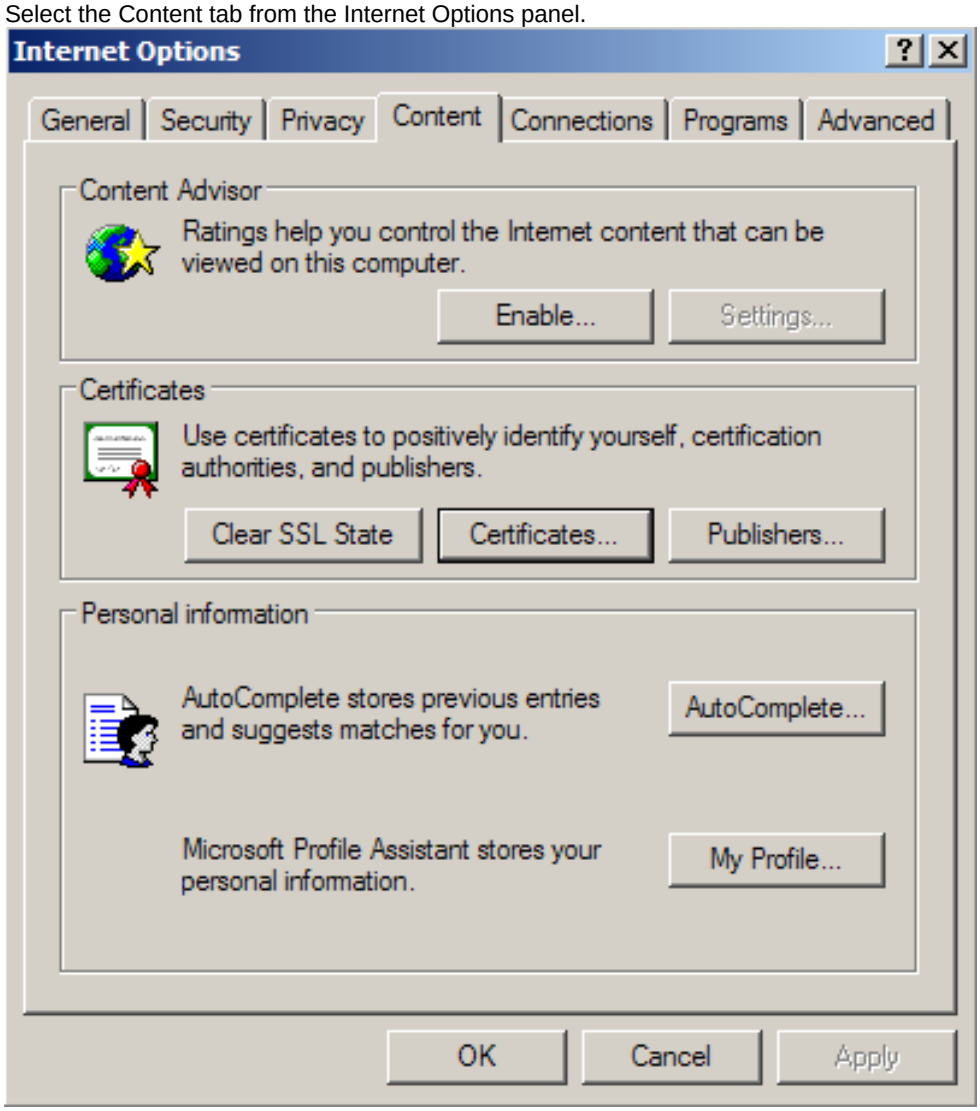
Where are the trust points installed?

In addition to the TrustedRoots property setting, the trust points also need to be installed in the Windows trusted root certificate store. You can do this by importing the DoD trusted root certificate by using Internet Explorer.

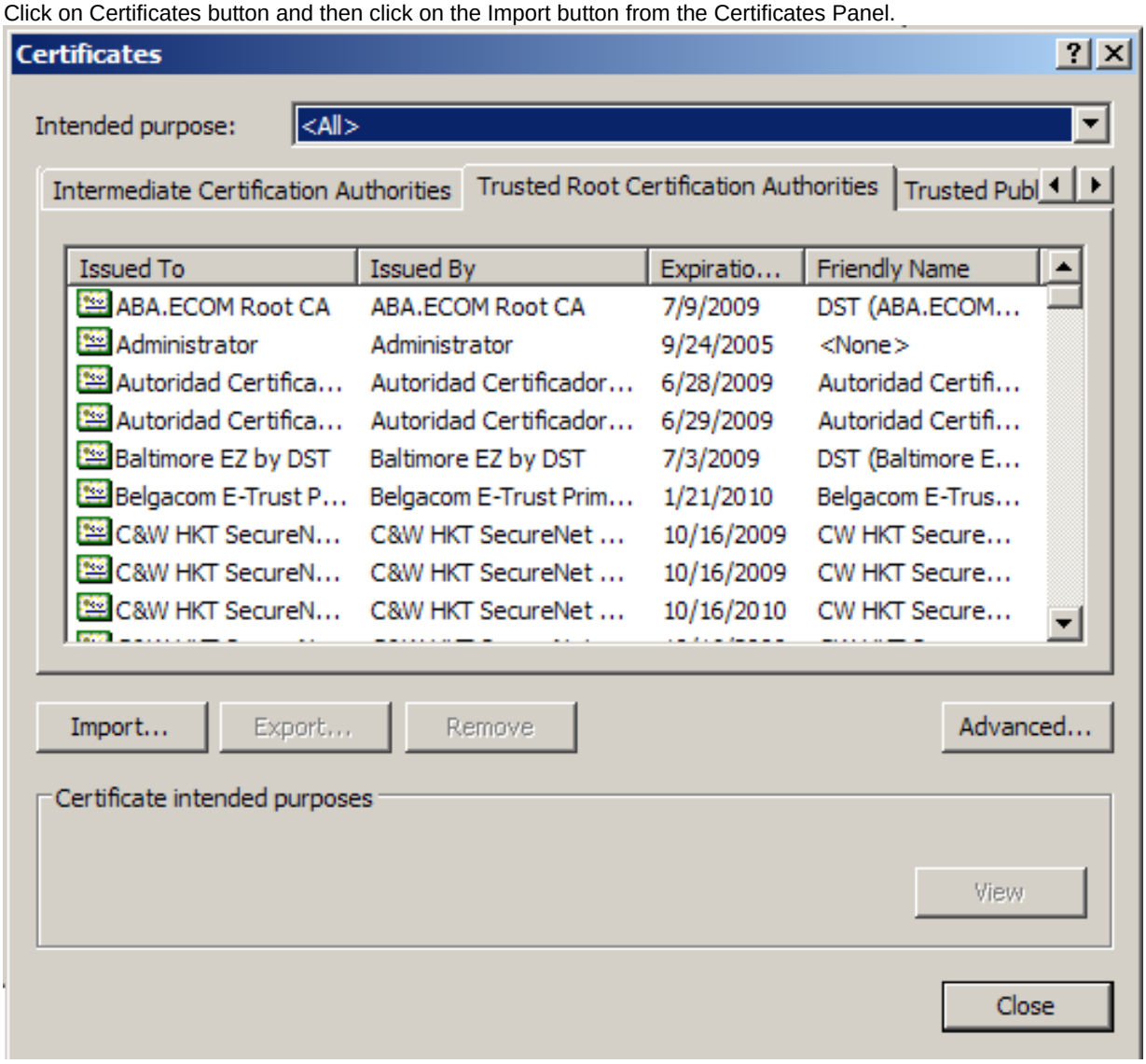
The following images show how you can import the DoD Trust Points using Internet Explorer:
Select Tools->Internet Options from the menu bar.



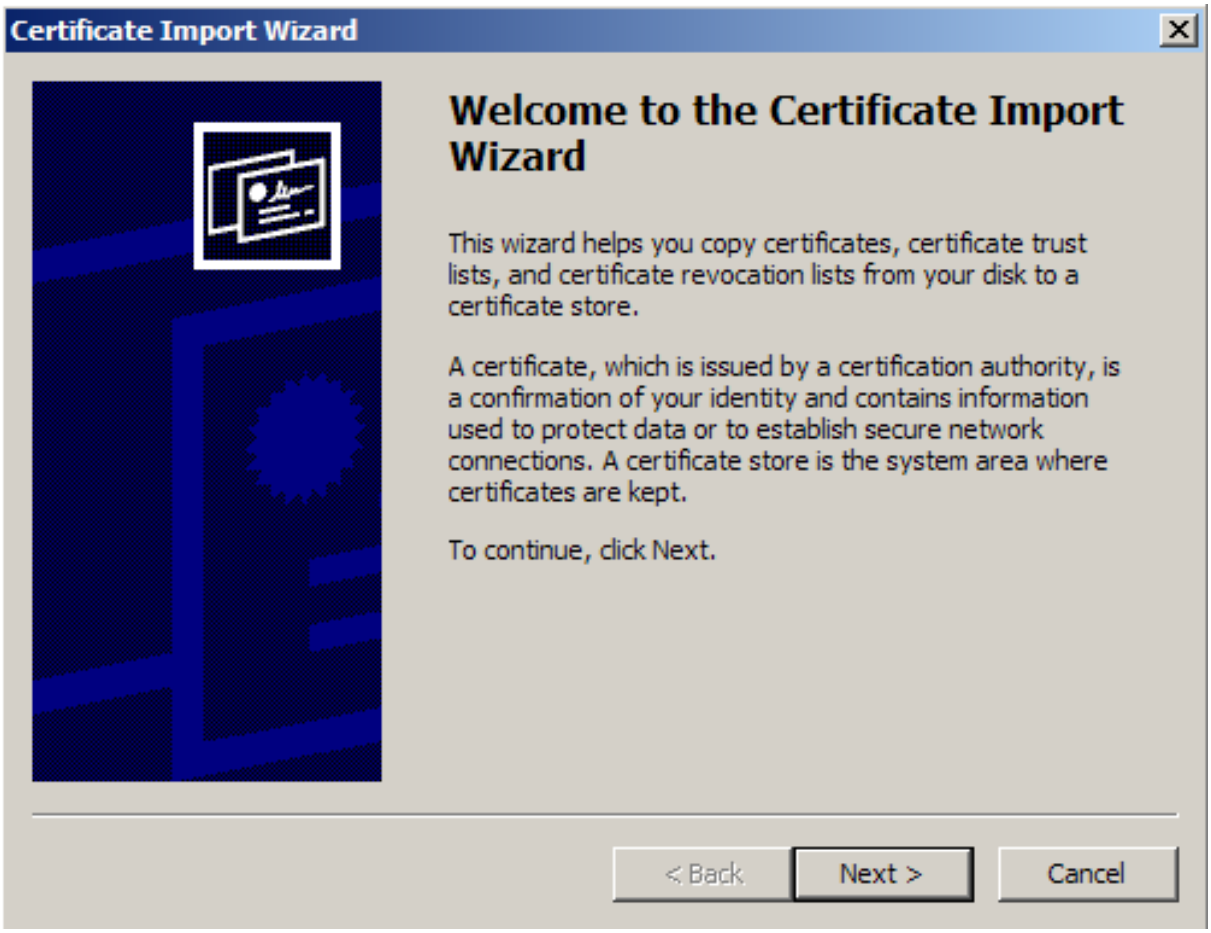
Select the Content tab from the Internet Options panel.



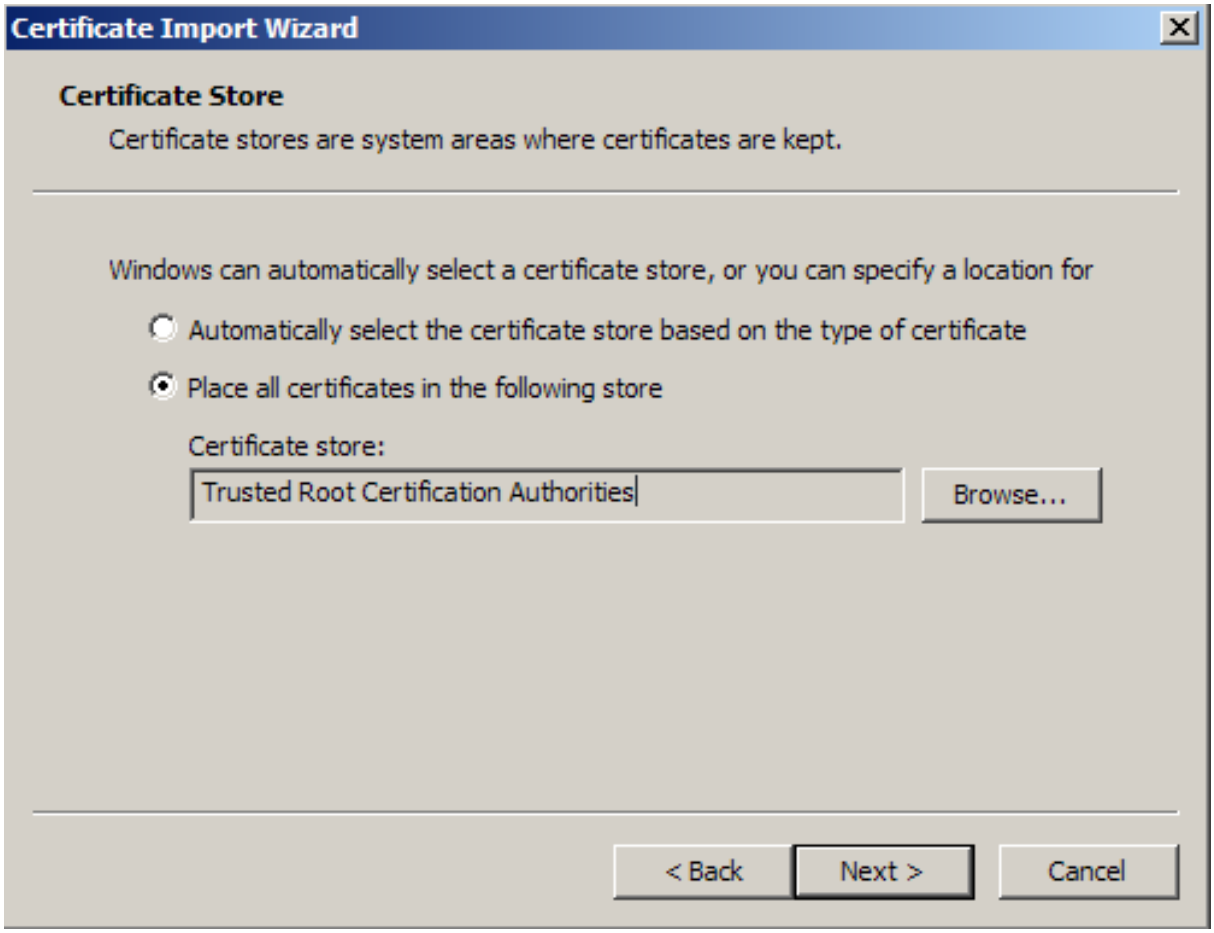
Click on Certificates button and then click on the Import button from the Certificates Panel.



Certificate import wizard will start. Click on the Next button and then click on Browse button to select your DoD Trusted Root certificate.



Select the *Place all certificate in the following store* radio button and click on the Browse button to select the "Trusted Root Certification Authorities" certificate store. Click on Next and then click on Finish. You should see a certificate imported successfully message.



Removing non-DOD PKI trust points

Setting CertficateTrustExplicit or DoDCompliance to 1 automatically disables all root certificates except those pointed to by TrustedRoots property. A DoD PKI Compliant Application will simply set DoDComplance to 1 and set TrustedRoots to the right root certificate file.

Importing keys and certificates

There are three ways to use certificates with SecureXML

1. Import your certificate into Windows Personal Store and then either let SecureXML provide a GUI for selecting this certificate or select this certificate by calling SetActiveCertificate() method programmatically. The procedure for importing the certificates into Windows Personal Certificate Store is the same as the one described for importing Trust Points above except the name of the certificate store to which to import the certificate to, is “Personal” instead of “Trusted Root Certification Authorities” in the last stage.

2. Specify a PFX/P12 file containing user public and private keys and provide a password for using the private key.

3. Provide a base64 encoded certificate to SecureXML by setting SignerCertificate property. The base64 encoded certificate contains only the public key. When the application invokes a signature creation method, SecureXML will access appropriate certificate store or hardware device such as CAC for accessing the

private key for completing the signature creation process. If LDAP server is used to store base64 encoded certificates, you can use ReadAll() method to fetch the certificates and if the LDAP certificate stores DER encoded certificates you can use ReadAllBase64() method to fetch the certificate and base64 encode it.

Installing Uniform Resource Indicators for DOD PKI services

SecureXML first looks at the signer's certificate for a CRL distribution point. If a CRL distribution point is found, SecureXML will access it using either http or LDAP protocols depending on the access method described in the CRL distribution point entry in the certificate. If a CRL distribution point is not found, SecureXML looks at the locations pointed to by the CRLLocation property set by the application. Each of the locations set by CRLLocation property can be a local or network file, a web file accessible via http or ldap protocols. No special configuration is needed.

Configuring (APPLICATION) properly to be interoperable with the DOD PKI according to DOD requirements

In order to fully comply with DoD PKI and be interoperable with DoD PKI, the application must set DoDCompliance to 1 and set appropriate certificate policies by setting the CertificatePolicy property to the required set of certificate policies.