XAdESSigner Component

Properties Methods Events Config Settings Errors

The XAdESSigner component creates XAdES-compliant signature files.

Syntax

```
nsoftware.SecureBlackbox.XAdESSigner
```

Remarks

XAdESSigner can sign XML documents in accordance with XAdES standard. Originally developed and adopted in the European Union, XAdES has quickly become a recognized international standard for signing XML documents. XAdES provides a convenient framework for creating short-lived and long-term signatures over any kind of XML documents, and is now used by governments, healthcare providers, banks, and independent service providers all across the globe.

Copy

```
XAdESSigner signer = new XAdESSigner();

signer.getNewSignature().setHashAlgorithm("SHA512"); // Default hashing algorithm SHA256

signer.getNewSignature().setLevel(XAdESSignature.aslBaselineT); // Level of the signature

// Select the document which contains the file that will be signed
signer.setInputFile("MyFile.xml");

// Select the path where you want to save the signed document
signer.setOutputFile("signedFile.xml");

// The certificate which the document will be signed with
signer.setSigningCertificate(new Certificate("test.pfx","test"));

// Provide the address of the Time Stamping Authority (TSA) server to be used for timestamping
signer.setTimestampServer("http://time.certum.pl");

signer.sign();
```

Standards and technologies supported

XAdESSigner offers the following signing capabilities:

- Create and upgrade XAdES signatures in accordance with the most recent XAdES specification (ETSI EN 319 132). Earlier versions are also supported.
- All profiles are supported (BES, EPES, T, C, X, XL, A, including the Extended variants).
- Timestamping using external TSAs.
- All industry-standard cryptographic algorithms (RSA, ECDSA, SHA256-512, and many others).

Configuring the signature parameters

Configuring XAdESSigner to make it produce a signature of the right type is the main task you would need to perform in your code. Normally the service or software you will be communicating your XML documents to will provide you with the list of requirements that your signatures should match.

Typically, those will dictate the following key aspects of the signatures:

- The signature Form, sometimes referred to as Level (such BES, T, XL, A, or Extended-XLong). This can be adjusted with the Level property.
- Whether the signature should be detached, enveloped, or enveloping: adjust via SignatureType property.
- When creating a timestamped signature (such as T or A), provide the address of your online TSA service via TimestampServer property.
- When creating long-term signatures that include the signing chain and validation material, tune up validation parameters via RevocationCheck, OfflineMode, and IgnoreChainValidationErrors properties.

In some circumstances you will also need to adjust the following lower-level settings:

- Choose a specific XAdES version between 1.4.1, 1.3.2, 1.2.2, and 1.1.1, and assign it to XAdESVersion property.
- Specify the needed canonicalization method using the CanonicalizationMethod property.
- Provide the hash algorithm via the HashAlgorithm property.

Signing certificates

XAdESSigner can use certificates residing on different media. Besides generic certificates stored in PFX or PEM files (A1), it can operate with non-exportable certificates residing on hardware media (A3) or in the cloud.

Non-exportable certificates can be accessed transparently via a Windows CSP or a PKCS#11 driver, if supplied by the certificate issuer. Proprietary interfaces can be plugged in with the *external signing* feature (see below).

You can use CertificateManager and CertificateStorage components to access the signing certificate. Assign the certificate to SigningCertificate property, and optionally provide the remainder of its chain via SigningChain property.

Note: If signing with a non-exportable key (such as residing on a hardware device or in the cloud), please make sure you keep the original CertificateStorage object open until the signing is completed. This is because the storage component provides a 'bridge' to the private key. If the storage is closed prematurely, this bridge is destroyed, and the private key can't be used.

You don't need to provide a signing certificate or chain when timestamping and upgrading signatures, since this type of operation does not involve the signing private key.

Signing the document

Now that you have set up all signature properties and attached the signing certificate, it is time to proceed to signing. You can provide the input document in one of the following forms: as a file (assign the path to InputFile property), as a stream (assign to InputStream property), or as a byte array (assign to InputBytes). Similarly, the output can be collected in one of the same forms, either by passing the destination path or stream via OutputFile and OutputStream respectively, or by reading the resulting document bytes from the OutputBytes property after the signing completes.

Having set up the input and output (unless using OutputBytes, which should be read later), call the component's Sign method. This will initiate the signing process. Depending on the settings, the signing may be as straightforward as calculating the document hash and signing it with the private key (e.g. in XAdES-BES variant), or it may involve advanced chain validation routines (XAdES-XL or -A). During the latter the component may contact a number of external revocation information sources (CRL and OCSP servers) to establish the validity of the signing certificate.

If a TSA server was provided via the TimestampServer property, the component will contact it too to timestamp the new signature.

During the signing XAdESSigner may fire events to let your code know of certain conditions. It may fire TLSCertValidate if one of the HTTP endpoints involved in the operation (which may be a CRL, OCSP, or TSA service) works over TLS and needs its certificate to be validated. It may also fire FormatElement and FormatText to let your code apply custom formatting to XML document elements. If XAdESSigner fails to resolve one of the references in the signature, it will fire ResolveReference to let your code help with resolving it.

When the signing operation completes, the output file, stream, or byte array will contain the signature of the requested kind. Note that while nveloped and enveloping signatures contain the signed content within them, detached signatures assume that you supply the original content separately.

Apart from signing, XAdESSigner can perform operations on signatures of other kinds. Use Upgrade method to upgrade an existing XAdES signature to a higher level (e.g. BES to XL). Use Timestamp to add a generic or validation timestamp to an existing signature.

External signing and DCAuth

XAdESSigner, like many other components offered by the product, supports two methods of signing with external keys. These methods are fully independent of each other: you can choose the one that suits your usage scenario best.

Synchronous method: ExternalSign

This is a simpler method that basically lets you infiltrate into the heart of the signing routine by taking care of the hash signing operation. The component does the rest of the job (hash calculation, preparation of signature objects, CRL/OCSP retrieval).

To initiate this method, call SignExternal instead of Sign. When the hash is ready, it will be passed back to your code with ExternalSign event. Your event handler needs to sign the hash with the private key and return the created signature back to the component - which will embed it into the document.

You don't need your signing certificate to contain an associated private key when using this method. The certificate itself (its public copy) may be needed though, as it is often included in the hash calculation.

This method is synchronous, meaning SignExternal provides you the results immediately upon its completion.

Asynchronous method: DCAuth

DCAuth is a SecureBlackbox-own know-how technology. This protocol was designed to allow sharing of private keys across environments, allowing the signer and the private key to reside on different systems. It works in the following way:

- The signing party - such as XAdESSigner - initiates the operation using SignAsyncBegin call. This produces two outcomes: a *pre-signed document* (a document with a blank signature placeholder), and a *request state* (an object containing a hash that needs to be signed). At this point the XAdESSigner instance can be released, and the process itself terminated (which may be useful when run as part of a web page).
- The *request state* is passed to the private key holder party. The private key holder passes the request state to a DCAuth object, which parses the request state, extracts the hash, and signs it. The output of DCAuth processing is another object, *response state*, which contains the signature. The private key holder then sends the *response state* back to the signing party.
- The signing party re-creates the controls, and passes the *response state*, together with the pre-signed version of the document, to the signer's SignAsyncEnd method. SignAsyncEnd extracts the signature from the *response state* and incorporates it into the pre-signed document.

This method is asynchronous in that sense that, from the signing party's viewpoint, it splits the signing operation into the *pre-signing* and *completion* stages which can be performed independently from each other and in different execution contexts. This makes this method particularly helpful for use in web pages and other scenarios where the signing key is not available in real time.

Fine-grained chain validation setup

Chain validation is a sophisticated, multi-faceted procedure that involves a lot of variables. Depending on the configuration of your operating environment, the specifics of the PKI framework being used, and the validation policy you need to follow, you may want to tune up your chain validation parameters so they fit them best. A summary of such parameters is given below.

- RevocationCheck lets you choose between and/or prioritize revocation origins. OCSP sources are often preferred to CRL because of their real-time capability and the smaller size of validation tokens they produce.
- OfflineMode is a master switch that stops the component from looking for any validation tokens online. If this property is switched on, the component will only use the KnownCertificates, TrustedCertificates, KnownCRLs, and KnownOCSPs collections to look for the missing validation material.
- IgnoreChainValidationErrors makes the component ignore any major validation issues it encounters (such us an untrusted chain or missing CRL). This option is handy for debugging and for creating signatures in the environments where the signing certificate is not trusted.
- KnownCertificates, KnownCRLs, and KnownOCSPs let you provide your own validation material. This may be useful when working in OfflineMode, where the signer has no access to the validation sources, or where the validation material has already been collected.
- TrustedCertificates lets you provide a list of trust anchors, either as a complement to the system's or as an alternative to it.
- BlockedCertificates lets you provide a list of blocked/distrusted certificates. Any CA certificate contained in it will be deemed untrusted/invalid.

The following parameters are not directly related to chain validation, but may have an implicit effect on it.

- Proxy, SocketSettings, and TLSSettings let you tune up the connectivity and TLS options in accordance with local preferences.
- TLSClientChain lets you provide the client certificate and its chain for TLS client authentication.
- Subscribe to TLSCertValidate to validate any TLS certificates of the services involved in chain validation.

The results of the chain validation procedure, upon its completion, are published in the following properties:

- ChainValidationResult contains the primary result of the chain validation routine: valid, valid but untrusted, invalid, or undefined.
- ChainValidationDetails provides the details of the factors that contributed to the chain validation result, such as an outdated certificate, a missing CRL, or a missing CA certificate.
- ValidationLog contains the detailed chain validation log. The log can often be very helpful in nailing down various validation issues.

Property List

The following is the full list of the properties of the component with short descriptions. Click on the links for further details.

AutoValidateSignatures	Specifies whether component should validate any present signatures when the document is opened.
BlockedCertificates	The certificates that must be rejected as trust anchors.
CheckTrustedLists	Specifies whether the component should attempt to validate chain trust via a known Trusted List.
CRLs	A collection of certificate revocation lists embedded into the signature by the signer.
DataBytes	Use this property to pass the external data to component in the byte array form.
DataFile	A file containing the external data covered by a detached signature.
DataStream	Standalone data covered by the signature.
DataType	Specifies the external data type.
DataURI	Specifies a detached data resource URI.
Encoding	Specifies XML encoding.
ExternalCrypto	Provides access to external signing and DC parameters.
FIPSMode	Reserved.
IgnoreChainValidationErrors	Makes the component tolerant to chain validation errors.
InputBytes	Use this property to pass the input to component in byte array form.
InputFile	The XML document to sign.
InputStream	A stream containing the document to sign.
KnownCertificates	Additional certificates for chain validation.
KnownCRLs	Additional CRLs for chain validation.
KnownOCSPs	Additional OCSP responses for chain validation.
NewSignature	Provides access to new signature properties.
OCSPs	A collection of OCSP responses embedded into the signature.
OfflineMode	Switches the component to offline mode.

OutputBytes	Use this property to read the output the component object has produced.
OutputFile	Specifies the file where the signed document will be saved.
OutputStream	The stream where the signed document is saved.
Profile	Specifies a pre-defined profile to apply when creating the signature.
Proxy	The proxy server settings.
References	A list of references to the data to be signed.
RevocationCheck	Specifies the kind(s) of revocation check to perform for all chain certificates.
Signatures	Provides details of all signatures found in the XML document.
SigningCertificate	The certificate to be used for signing.
SigningChain	The signing certificate chain.
SocketSettings	Manages network connection settings.
Timestamps	Contains a collection of timestamps for the processed document.
TimestampServer	The address of the timestamping server.
TLSClientChain	The TLS client certificate chain.
TLSServerChain	The TLS server's certificate chain.
TLSSettings	Manages TLS layer settings.
TrustedCertificates	A list of trusted certificates for chain validation.
ValidationMoment	The time point at which signature validity is to be established.

Method List

The following is the full list of the methods of the component with short descriptions. Click on the links for further details.

AddDataReference	Creates a new XML reference to the specified data.
AddKnownNamespace	Adds known prefix and correspondent namespace URI.
AddReference	Creates a new XML reference to the specified XML element.
Close	Closes an opened document.
Config	Sets or retrieves a configuration setting.

DoAction	Performs an additional action.
ExtractAsyncData	Extracts user data from the DC signing service response.
GetInnerXML	Get the inner XML content of the selected XML element.
GetOuterXML	Get the outer XML content of the selected XML element.
GetTextContent	Get the text content of the selected XML element.
Open	Opens a document for signing or updating.
Reset	Resets the component settings.

SetInnerXML	Set the inner XML content of the selected XML element.
SetTextContent	Set the text content of the selected XML element.
Sign	Signs an XML document.
SignAsyncBegin	Initiates the asynchronous signing operation.
SignAsyncEnd	Completes the asynchronous signing operation.
SignExternal	Signs the document using an external signing facility.
Timestamp	Use this method to add an timestamp.
Upgrade	Upgrades existing XAdES signature to a new form.

Event List

The following is the full list of the events fired by the component with short descriptions. Click on the links for further details.

ChainElementDownload	Fires when there is a need to download a chain element from an online source.
ChainElementNeeded	Fires when an element required to validate the chain was not located.
ChainElementStore	This event is fired when a chain element (certificate, CRL, or OCSP response) should be stored along with a signature.
ChainValidated	Reports the completion of a certificate chain validation.
ChainValidationProgress	This event is fired multiple times during chain validation to report various stages of the validation procedure.

DocumentLoaded	This event is fired when the document has been loaded into memory.
Error	Information about errors during signing.
ExternalSign	Handles remote or external signing initiated by the SignExternal method or other source.
FormatElement	Reports the XML element that is currently being processed.
FormatText	Reports XML text that is currently being processed.
Notification	This event notifies the application about an underlying control flow event.
ReferenceValidated	Marks the end of a reference validation.
ResolveReference	Asks the application to resolve a reference.
SignatureFound	Signifies the start of signature validation.
SignatureValidated	Marks the completion of the signature validation routine.
TimestampFound	Signifies the start of a timestamp validation routine.
TimestampRequest	Fires when the component is ready to request a timestamp from an external TSA.
TimestampValidated	Reports the completion of the timestamp validation routine.
TLSCertNeeded	Fires when a remote TLS party requests a client certificate.
TLSCertValidate	This event is fired upon receipt of the TLS server's certificate, allowing the user to control its acceptance.
TLSEstablished	Fires when a TLS handshake with *Host* successfully completes.
TLSHandshake	Fires when a new TLS handshake is initiated, before the handshake commences.
TLSShutdown	Reports the graceful closure of a TLS connection.

Config Settings

The following is a list of config settings for the component with short descriptions. Click on the links for further details.

AddAllDataObjectsTimestamp	Whether to add all data objects timestamp during signing.
ClaimedRolesXML	The XML content of the claimed roles.

ClaimedRoleText	The text of the claimed role.
CommitmentTypeIndicationAllSignedDataObjects[Index]	Specifies the CommitmentTypeIndication's AllSignedDataObjects.
CommitmentTypeIndicationCount	The number of the CommitmentTypeIndication elements.
CommitmentTypeIndicationIdentifier[Index]	Specifies the CommitmentTypeIndication's CommitmentTypeId's Identifier.
CommitmentTypeIndicationIdentifierDescription[Index]	Specifies the CommitmentTypeIndication's CommitmentTypeId's Description.
CommitmentTypeIndicationIdentifierDocumentationReferences[Index]	Specifies the CommitmentTypeIndication's CommitmentTypeId's DocumentationReferences.
CommitmentTypeIndicationIdentifierQualifier[Index]	Specifies the CommitmentTypeIndication's CommitmentTypeId's IdentifierQualifier.
CommitmentTypeIndicationObjectReference[Index]	Specifies the CommitmentTypeIndication's ObjectReference.
CommitmentTypeIndicationQualifiersXML[Index]	The XML content of the CommitmentTypeIndication's Qualifiers.
DataObjectFormatCount	The number of the DataObjectFormat elements.
DataObjectFormatDescription[Index]	Specifies the DataObjectFormat's Description.
DataObjectFormatEncoding[Index]	Specifies the DataObjectFormat's Encoding.
DataObjectFormatMimeType[Index]	Specifies the DataObjectFormat's MimeType.

DataObjectFormatObjectIdentifier[Index]	Specifies the DataObjectFormat's ObjectIdentifier's Identifier.
DataObjectFormatObjectIdentifierDescription[Index]	Specifies the DataObjectFormat's ObjectIdentifier's Description.
DataObjectFormatObjectIdentifierDocumentationReferences[Index]	Specifies the DataObjectFormat's ObjectIdentifier's DocumentationReferences.
DataObjectFormatObjectIdentifierQualifier[Index]	Specifies the DataObjectFormat's ObjectIdentifier's IdentifierQualifier.
DataObjectFormatObjectReference[Index]	Specifies the DataObjectFormat's ObjectReference.
DataType	Specifies the external data type.
DetachedResourceURI	Specifies a detached resource URI.
EnvelopingObjectEncoding	Specifies the enveloping object encoding.
EnvelopingObjectID	Specifies the enveloping object identifier.
EnvelopingObjectMimeType	Specifies the enveloping object MIME type.
ExclusiveCanonicalizationPrefix	Specifies the exclusive canonicalization prefix.
HMACKey	The key value for HMAC.
HMACOutputLength	Sets the length of the HMAC output.
HMACSigningUsed	Whether to use HMAC signing.
IDAttributeName	Specifies the custom name of ID attribute.
IDAttributeNamespaceURI	Specifies the custom namespace URI of ID attribute.

IgnoreTimestampFailure	Whether to ignore time-stamping failure during signing.
IncludeKey	Specifies whether to include the signing key to the signature.
IncludeKeyValue	Specifies whether the key value must be included to the signature.
IncludeKnownRevocationInfoToSignature	Whether to include custom revocation info to the signature.
InclusiveNamespacesPrefixList	Specifies the InclusiveNamespaces PrefixList.
InputType	Specifies the Input type.
InsertBeforeXMLElement	Defines the reference XML element for signature insertion.
KeyInfoCustomXML	The custom XML content for KeyInfo element.
KeyInfoDetails	Specifies the signing key info details to include to the signature.
KeyInfoID	Specifies the ID for KeyInfo element.
KeyInfoX509Chain	Specifies which certificates from SigningChain are included in the ds:KeyInfo element.
KeyName	Contains information about the key used for signing.
ManifestCount	TBD.
ManifestID[i]	TBD.
ManifestObjectIndex[i]	TBD.
ManifestXML[i]	TBD.
NormalizeNewLine	Controls whether newline combinations should be automatically normalized.

ObjectCount	TBD.
ObjectEncoding[i]	TBD.
ObjectID[i]	TBD.
ObjectMimeType[i]	TBD.
ObjectSignaturePropertiesCount	TBD.
ObjectSignaturePropertiesID[i]	TBD.
ObjectSignaturePropertiesObjectIndex[i]	TBD.
ObjectSignaturePropertiesXML[i]	TBD.
ObjectSignaturePropertyCount	TBD.
ObjectSignaturePropertyID[i]	TBD.
ObjectSignaturePropertyPropertiesIndex[i]	TBD.
ObjectSignaturePropertyTarget[i]	TBD.
ObjectSignaturePropertyXML[i]	TBD.
ObjectXML[i]	TBD.
PolicyDescription	signature policy description.
PolicyExplicitText	The explicit text of the user notice.
PolicyUNNumbers	The noticeNumbers part of the NoticeReference CAdES attribute.
PolicyUNOrganization	The organization part of the NoticeReference qualifier.
ProductionPlace	Identifies the place of the signature production.
PSSUsed	Whether to use RSASSA-PSS algorithm.
QualifyingPropertiesID	Specifies the ID for QualifyingProperties element.
QualifyingPropertiesObjectID	Specifies the ID for object with QualifyingProperties element.
QualifyingPropertiesReferenceCount	The number of the QualifyingPropertiesReference elements.

QualifyingPropertiesReferenceID[Index]	Specifies the QualifyingPropertiesReference's ID.
QualifyingPropertiesReferenceURI[Index]	Specifies the QualifyingPropertiesReference's URI.
RefsTimestampType	Specifies references timestamp type to include to the signature.
SignatureCompliance	Specifies the signature compliance mode.
SignatureID	Specifies the ID for Signature element.
SignaturePrefix	Specifies the signature prefix.
SignatureValueID	Specifies the ID for SignatureValue element.
SignedInfoID	Specifies the ID for SignedInfo element.
SignedPropertiesID	Specifies the ID for SignedProperties element.
SignedPropertiesReferenceCanonicalizationMethod	Specifies the canonicalization method used in SignedProperties reference.
SignedPropertiesReferenceHashAlgorithm	Specifies the hash algorithm used in SignedProperties reference.
SignedPropertiesReferenceID	Specifies the ID for Reference element that points to SignedProperties element.
SignedPropertiesReferenceInclusiveNamespacesPrefixList	Specifies the InclusiveNamespaces PrefixList used in SignedProperties reference.
SignedPropertiesReferenceIndex	Specifies the index of SignedProperties reference.
SignedSignaturePropertiesID	Specifies the ID for SignedSignatureProperties

	element.
SigningCertificatesChain	Specifies which certificates from SigningChain are included in the xades:SigningCertificate element.
SigningCertificatesHashAlgorithm	Specifies the hash algorithm used for SigningCertificates.
SigningTimeFormat	Specifies the date time format for the XAdES signing time.
SigningTimeIsUTC	Specifies whether the XAdES signing time is in UTC.
SigningTimeZoneOffset	Specifies the time zone offset for the XAdES signing time.
SigPolicyDescription	signature policy description.
SigPolicyExplicitText	The explicit text of the user notice.
SigPolicyHash	The EPES policy hash.
SigPolicyHashAlgorithm	The hash algorithm that was used to generate the EPES policy hash.
SigPolicyID	The EPES policy ID.
SigPolicyNoticeNumbers	The noticeNumbers part of the NoticeReference CAdES attribute.
SigPolicyNoticeOrganization	The organization part of the NoticeReference qualifier.
SigPolicyURI	The EPES policy URI.
StripWhitespace	Controls whether excessive whitespace characters should be stripped off when loading the document.
TimestampCanonicalizationMethod	Specifies canonicalization method used in timestamp.
TimestampValidationDataDetails	Specifies timestamp validation data details to include to the signature.

UseCustomTransformOrder	Enables custom ordering for the document transforms.
UseHMACSigning	Whether to use HMAC signing.
UseMultipleX509DataNodes	Controls whether certificates are serialized in multiple ds:X509Data nodes under ds:KeyInfo.
UsePSS	Whether to use RSASSA-PSS algorithm.
ValidationDataRefsDetails	Specifies validation data references details to include to the signature.
ValidationDataRefsHashAlgorithm	Specifies the hash algorithm used in validation data references.
ValidationDataValuesDetails	Specifies validation data values details to include to the signature.
WriteBOM	Specifies whether byte-order mark should be written when saving the document.
XAdESPrefix	Specifies the XAdES prefix.
XAdESv141Prefix	Specifies the XAdES v1.4.1 prefix.
XMLFormatting	Specifies the signature XML formatting.
XMLSerializationCanonicalizationMethod	Specifies the canonicalization method to use for serialization.
XMLSerializationMode	Specifies the serialization mode for the extracted part of XML document.
ASN1UseGlobalTagCache	Controls whether ASN.1 module should use a global object cache.

AssignSystemSmartCardPins	Specifies whether CSP-level PINs should be assigned to CNG keys.
CheckKeyIntegrityBeforeUse	Enables or disable private key integrity check before use.
CookieCaching	Specifies whether a cookie cache should be used for HTTP(S) transports.
Cookies	Gets or sets local cookies for the component.
DefDeriveKeyIterations	Specifies the default key derivation algorithm iteration count.
DNSLocalSuffix	The suffix to assign for TLD names.
EnableClientSideSSLFFDHE	Enables or disables finite field DHE key exchange support in TLS clients.
GlobalCookies	Gets or sets global cookies for all the HTTP transports.
HardwareCryptoUsePolicy	The hardware crypto usage policy.
HttpUserAgent	Specifies the user agent name to be used by all HTTP clients.
HttpVersion	The HTTP version to use in any inner HTTP client components created.
IgnoreExpiredMSCTLSigningCert	Whether to tolerate the expired Windows Update signing certificate.
ListDelimiter	The delimiter character for multi-element lists.
LogDestination	Specifies the debug log destination.
LogDetails	Specifies the debug log details to dump.

LogFile	Specifies the debug log filename.
LogFilters	Specifies the debug log filters.
LogFlushMode	Specifies the log flush mode.
LogLevel	Specifies the debug log level.
LogMaxEventCount	Specifies the maximum number of events to cache before further action is taken.
LogRotationMode	Specifies the log rotation mode.
MaxASN1BufferLength	Specifies the maximal allowed length for ASN.1 primitive tag data.
MaxASN1TreeDepth	Specifies the maximal depth for processed ASN.1 trees.
OCSPHashAlgorithm	Specifies the hash algorithm to be used to identify certificates in OCSP requests.
OldClientSideRSAFallback	Specifies whether the SSH client should use a SHA1 fallback.
PKICache	Specifies which PKI elements (certificates, CRLs, OCSP responses) should be cached.
PKICachePath	Specifies the file system path where cached PKI data is stored.
ProductVersion	Returns the version of the SecureBlackbox library.
ServerSSLDHKeyLength	Sets the size of the TLS DHE key exchange group.
StaticDNS	Specifies whether static DNS rules should be used.
StaticIPAddress[domain]	Gets or sets an IP address for the specified domain name.

StaticIPAddresses	Gets or sets all the static DNS rules.
Tag	Allows to store any custom data.
TLSSessionGroup	Specifies the group name of TLS sessions to be used for session resumption.
TLSSessionLifetime	Specifies lifetime in seconds of the cached TLS session.
TLSSessionPurgeInterval	Specifies how often the session cache should remove the expired TLS sessions.
UseCRLObjectCaching	Specifies whether reuse of loaded CRL objects is enabled.
UseInternalRandom	Switches between SecureBlackbox-own and platform PRNGs.
UseLegacyAdESValidation	Enables legacy AdES validation mode.
UseOCSPResponseObjectCaching	Specifies whether reuse of loaded OCSP response objects is enabled.
UseOwnDNSResolver	Specifies whether the client components should use own DNS resolver.
UseSharedSystemStorages	Specifies whether the validation engine should use a global per-process copy of the system certificate stores.
UseSystemNativeSizeCalculation	An internal CryptoAPI access tweak.
UseSystemOAEPAndPSS	Enforces or disables the use of system-driven RSA OAEP and PSS computations.
UseSystemRandom	Enables or disables the use of the OS PRNG.

XMLRDNDescriptorName[OID]	Defines an OID mapping to descriptor names for the certificate's IssuerRDN or SubjectRDN.
XMLRDNDescriptorPriority[OID]	Specifies the priority of descriptor names associated with a specific OID.
XMLRDNDescriptorReverseOrder	Specifies whether to reverse the order of descriptors in RDN.
XMLRDNDescriptorSeparator	Specifies the separator used between descriptors in RDN.

AutoValidateSignatures Property (XAdESSigner Component)

Specifies whether component should validate any present signatures when the document is opened.

Syntax

C# VB.NET

```
public bool AutoValidateSignatures { get; set; }
```

Default Value

False

Remarks

This setting is switched off by default to speed up document processing. Even if the document is loaded with this property set to false, you can validate the signatures manually on a later stage using the Revalidate method.

BlockedCertificates Property (XAdESSigner Component)

The certificates that must be rejected as trust anchors.

Syntax

C# VB.NET

```
public CertificateList BlockedCertificates { get; }
```

Remarks

Use this property to provide a list of compromised or blocked certificates. Any chain containing a blocked certificate will fail validation.

This property is not available at design time.

Please refer to the Certificate type for a complete list of fields.

CheckTrustedLists Property (XAdESSigner Component)

Specifies whether the component should attempt to validate chain trust via a known Trusted List.

Syntax

C# VB.NET

```
public bool CheckTrustedLists { get; set; }
```

Default Value

False

Remarks

Set this property to true to enable the component to validate chain trust against an internal list of known Trusted Lists (such as EUTL).

CRLs Property (XAdESSigner Component)

A collection of certificate revocation lists embedded into the signature by the signer.

Syntax

C# VB.NET

```
public CRLList CRLs { get; }
```

Remarks

Use this property to access the CRLs embedded into the signature by the signer.

This property is read-only and not available at design time.

Please refer to the CRL type for a complete list of fields.

DataBytes Property (XAdESSigner Component)

Use this property to pass the external data to component in the byte array form.

Syntax

C# VB.NET

```
public byte[] DataBytes { get; set; }
```

Remarks

Assign a byte array containing the external data to be processed to this property.

This property is not available at design time.

DataFile Property (XAdESSigner Component)

A file containing the external data covered by a detached signature.

Syntax

C# VB.NET

```
public string DataFile { get; set; }
```

Default Value

""

Remarks

In the case of a detached signature, use this property to provide the external data to the component from a file. Alternatively, provide the data via DataStream.

DataStream Property (XAdESSigner Component)

Standalone data covered by the signature.

Syntax

C# VB.NET

```
public System.IO.Stream DataStream { get; set; }
```

Default Value

null

Remarks

When validating a detached signature, use this stream property to provide the external (detached) data.

This property is not available at design time.

DataType Property (XAdESSigner Component)

Specifies the external data type.

Syntax

C# VB.NET

```
public XAdESSignerDataTypes DataType { get; set; }

enum XAdESSignerDataTypes {
  cxdtXML,
  cxdtBinary,
  cxdtBase64
}
```

Default Value

0

Remarks

Use this property to specify the type of the external data (either DataFile, DataStream or DataBytes properties) for component.

DataURI Property (XAdESSigner Component)

Specifies a detached data resource URI.

Syntax

C# VB.NET

```
public string DataURI { get; set; }
```

Default Value

""

Remarks

Specifies a URI used for data being signed, usually the data filename if stored along with a detached signature.

Encoding Property (XAdESSigner Component)

Specifies XML encoding.

Syntax

C# VB.NET

```
public string Encoding { get; set; }
```

Default Value

""

Remarks

Use this property to specify the encoding to apply to the XML documents.

ExternalCrypto Property (XAdESSigner Component)

Provides access to external signing and DC parameters.

Syntax

C# VB.NET

```
public ExternalCrypto ExternalCrypto { get; }
```

Remarks

Use this property to tune-up remote cryptography settings. SecureBlackbox supports two independent types of external cryptography: synchronous (based on the ExternalSign event) and asynchronous (based on the DC protocol and the DCAuth signing component).

This property is read-only.

Please refer to the ExternalCrypto type for a complete list of fields.

FIPSMode Property (XAdESSigner Component)

Reserved.

Syntax

C# VB.NET

```
public bool FIPSMode { get; set; }
```

Default Value

False

Remarks

This property is reserved for future use.

IgnoreChainValidationErrors Property (XAdESSigner Component)

Makes the component tolerant to chain validation errors.

Syntax

C# VB.NET

```
public bool IgnoreChainValidationErrors { get; set; }
```

Default Value

False

Remarks

If this property is set to True, any errors emerging during certificate chain validation will be ignored. This setting may be handy if the purpose of validation is the creation of an LTV signature, and the validation is performed in an environment that doesn't trust the signer's certificate chain.

InputBytes Property (XAdESSigner Component)

Use this property to pass the input to component in byte array form.

Syntax

C# VB.NET

```
public byte[] InputBytes { get; set; }
```

Remarks

Assign a byte array containing the data to be processed to this property.

This property is not available at design time.

InputFile Property (XAdESSigner Component)

The XML document to sign.

Syntax

C# VB.NET

```
public string InputFile { get; set; }
```

Default Value

""

Remarks

Provide a path to the XML file to sign.

InputStream Property (XAdESSigner Component)

A stream containing the document to sign.

Syntax

C# VB.NET

```
public System.IO.Stream InputStream { get; set; }
```

Default Value

null

Remarks

Use this property to pass the document to the component in a stream.

This property is not available at design time.

KnownCertificates Property (XAdESSigner Component)

Additional certificates for chain validation.

Syntax

C# VB.NET

```
public CertificateList KnownCertificates { get; }
```

Remarks

Use this property to supply a list of additional certificates that might be needed for chain validation. An example of a scenario where you might want to do that is when intermediary CA certificates are absent from the standard system locations (or when there are no standard system locations), and therefore should be supplied to the component manually.

The purpose of the certificates to be added to this collection is roughly equivalent to that of the Intermediate Certification Authorities system store in Windows.

Do not add trust anchors or root certificates to this collection: add them to TrustedCertificates instead.

This property is not available at design time.

Please refer to the Certificate type for a complete list of fields.

KnownCRLs Property (XAdESSigner Component)

Additional CRLs for chain validation.

Syntax

C# VB.NET

```
public CRLList KnownCRLs { get; }
```

Remarks

Use this property to supply additional CRLs that might be needed for chain validation. This property may be helpful when a chain is validated in offline mode, and the associated CRLs are stored separately from the signed message or document.

This property is not available at design time.

Please refer to the CRL type for a complete list of fields.

KnownOCSPs Property (XAdESSigner Component)

Additional OCSP responses for chain validation.

Syntax

C# VB.NET

```
public OCSPResponseList KnownOCSPs { get; }
```

Remarks

Use this property to supply additional OCSP responses that might be needed for chain validation. This property may be helpful when a chain is validated in offline mode, and the associated OCSP responses are stored separately from the signed message or document.

This property is not available at design time.

Please refer to the OCSPResponse type for a complete list of fields.

NewSignature Property (XAdESSigner Component)

Provides access to new signature properties.

Syntax

C# VB.NET

```
public XAdESSignature NewSignature { get; }
```

Remarks

Use this property to tune-up signature properties.

This property is read-only and not available at design time.

Please refer to the XAdESSignature type for a complete list of fields.

OCSPs Property (XAdESSigner Component)

A collection of OCSP responses embedded into the signature.

Syntax

C# VB.NET

```
public OCSPResponseList OCSPs { get; }
```

Remarks

Use this property to access the OCSP responses embedded into the signature by its creator.

This property is read-only and not available at design time.

Please refer to the OCSPResponse type for a complete list of fields.

OfflineMode Property (XAdESSigner Component)

Switches the component to offline mode.

Syntax

C# VB.NET

```
public bool OfflineMode { get; set; }
```

Default Value

False

Remarks

When working in offline mode, the component restricts itself from using any online revocation information sources, such as CRL or OCSP responders.

Offline mode may be useful if there is a need to verify the completeness of the validation information included within the signature or provided via KnownCertificates, KnownCRLs, and other related properties.

OutputBytes Property (XAdESSigner Component)

Use this property to read the output the component object has produced.

Syntax

C# VB.NET

```
public byte[] OutputBytes { get; }
```

Remarks

Read the contents of this property after the operation has completed to read the produced output. This property will only be set if the OutputFile and OutputStream properties had not been assigned.

This property is read-only and not available at design time.

OutputFile Property (XAdESSigner Component)

Specifies the file where the signed document will be saved.

Syntax

C# VB.NET

```
public string OutputFile { get; set; }
```

Default Value

""

Remarks

Provide the full path to the file where the signed document should be saved.

OutputStream Property (XAdESSigner Component)

The stream where the signed document is saved.

Syntax

C# VB.NET

```
public System.IO.Stream OutputStream { get; set; }
```

Default Value

null

Remarks

Use this property to specify the stream to save the signed document.

This property is not available at design time.

Profile Property (XAdESSigner Component)

Specifies a pre-defined profile to apply when creating the signature.

Syntax

C# VB.NET

```
public string Profile { get; set; }
```

Default Value

""

Remarks

Advanced signatures come in many variants, which are often defined by parties that needs to process them or by local standards. SecureBlackbox profiles are sets of pre-defined configurations which correspond to particular signature variants. By specifying a profile, you are pre-configuring the component to make it produce the signature that matches the configuration corresponding to that profile.

Supported profiles:

"ES.Factura"	Spanish Factura Electronica
"BR.AD_RB_v2_3"	Brazilian signature with Basic Reference (AD-RB) version 2.3
"BR.AD_RB_v2_4"	Brazilian signature with Basic Reference (AD-RB) version 2.4
"BR.AD_RT_v2_3"	Brazilian signature with Time Reference (AD-RT) version 2.3
"BR.AD_RT_v2_4"	Brazilian signature with Time Reference (AD-RT) version 2.4
"BR.AD_RV_v2_3"	Brazilian signature with References for Validation (AD-RV) version 2.3
"BR.AD_RV_v2_4"	Brazilian signature with References for Validation (AD-RV) version 2.4
"BR.AD_RC_v2_3"	Brazilian signature with Complete References (AD-RC) version 2.3
"BR.AD_RC_v2_4"	Brazilian signature with Complete References (AD-RC) version 2.4
"BR.AD_RA_v2_3"	Brazilian signature with References for Archiving (AD-RA) version 2.3
"BR.AD_RA_v2_4"	Brazilian signature with References for Archiving (AD-RA) version 2.4
"XAdES.CounterSignature"	Adds XAdES countersignature (supported in XML and XAdES signer control)

"ES.Factura" profile equivalent to the following settings:

Copy

```
using (XMLReference Ref = new XMLReference())
{
  Ref.ID = "Reference-1";
  Ref.TargetXMLElement = "";
  Ref.UseEnvelopedSignatureTransform = true;
  XAdESSigner.References.Add(Ref);
}

using (XMLReference Ref = new XMLReference())
{
  Ref.URI = "#KeyInfo-1";
  XAdESSigner.References.Add(Ref);
}

XAdESSigner.NewSignature.XAdESVersion = XAdESVersions.xav132;
XAdESSigner.NewSignature.Level = AdESSignatureLevels.aslEPES;
XAdESSigner.Config("KeyInfoDetails=certificate");
XAdESSigner.Config("KeyInfoID=KeyInfo-1");
```

```
XAdESSigner.Config("ClaimedRoleText=emisor");
XAdESSigner.Config("SigPolicyId=http://www.facturae.es/politica_de_firma_formato_facturae/poli
XAdESSigner.Config("SigPolicyDescription=Pol" + "\u00ED" + "tica de Firma FacturaE v3.1");
XAdESSigner.Config("SigPolicyHash=3a18b197aba90fa6aff0dee912f0c006110bea13");
XAdESSigner.Config("SigPolicyHashAlgorithm=SHA1");
XAdESSigner.Config("DataObjectFormatObjectReference=#Reference-1");
XAdESSigner.Config("DataObjectFormatMimeType=text/xml");
XAdESSigner.Config("DataObjectFormatDescription=Factura electr" + "\u00F3" + "nica");
```

"XAdES.CounterSignature" profile allows to add XAdES countersignature to the existent XAdES-BES or EPES signature. Use XMLElement property to specify existent XAdES signature. Sample code:

Copy

```
XAdESSigner.Profile = "XAdES.CounterSignature";
XAdESSigner.NewSignature.XAdES = false;
```

Proxy Property (XAdESSigner Component)

The proxy server settings.

Syntax

C# VB.NET

```
public ProxySettings Proxy { get; }
```

Remarks

Use this property to tune up the proxy server settings.

This property is read-only.

Please refer to the ProxySettings type for a complete list of fields.

References Property (XAdESSigner Component)

A list of references to the data to be signed.

Syntax

C# VB.NET

```
public XMLReferenceList References { get; }
```

Remarks

Electronic signature is computed over a set of data pieces. Each piece of data to be signed is specified by a reference.

This property is read-only and not available at design time.

Please refer to the XMLReference type for a complete list of fields.

RevocationCheck Property (XAdESSigner Component)

Specifies the kind(s) of revocation check to perform for all chain certificates.

Syntax

C# VB.NET

```
public XAdESSignerRevocationChecks RevocationCheck { get; set; }

enum XAdESSignerRevocationChecks {
  crcNone,
  crcAuto,
  crcAllCRL,
  crcAllOCSP,
  crcAllCRLAndOCSP,
  crcAnyCRL,
  crcAnyOCSP,
  crcAnyCRLOrOCSP,
  crcAnyOCSPOrCRL
}
```

Default Value

1

Remarks

Revocation checking is necessary to ensure the integrity of the chain and obtain up-to-date certificate validity and trustworthiness information.

Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP) responses serve the same purpose of ensuring that the certificate had not been revoked by the Certificate Authority (CA) at the time of use. Depending on your circumstances and security policy requirements, you may want to use either one or both of the revocation information source types.

crcNone	`0`	No revocation checking.
crcAuto	`1`	Automatic mode selection. Currently this maps to crcAnyOCSPOrCRL, but it may change in the future.
crcAllCRL	`2`	All provided CRL endpoints will be checked, and all checks must succeed.

		succeed.
crcAllCRLAndOCSP	`4`	All provided CRL and OCSP endpoints will be checked, and all checks must succeed.
crcAnyCRL	`5`	All provided CRL endpoints will be checked, and at least one check must succeed.
crcAnyOCSP	`6`	All provided OCSP endpoints will be checked, and at least one check must succeed.
crcAnyCRLOrOCSP	`7`	All provided CRL and OCSP endpoints will be checked, and at least one check must succeed. CRL endpoints are checked first.
crcAnyOCSPOrCRL	`8`	All provided CRL and OCSP endpoints will be checked, and at least one check must succeed. OCSP endpoints are checked first.

This setting controls the way the revocation checks are performed for every certificate in the chain. Typically certificates come with two types of revocation information sources: CRL (certificate revocation lists) and OCSP responders. CRLs are static objects periodically published by the CA at some online location. OCSP responders are active online services maintained by the CA that can provide up-to-date information on certificate statuses in near real time.

There are some conceptual differences between the two. CRLs are normally larger in size. Their use involves some latency because there is normally some delay between the time when a certificate was

revoked and the time the subsequent CRL mentioning that is published. The benefits of CRL is that the same object can provide statuses for all certificates issued by a particular CA, and that the whole technology is much simpler than OCSP (and thus is supported by more CAs).

This setting lets you adjust the validation course by including or excluding certain types of revocation sources from the validation process. The crcAnyOCSPOrCRL setting (give preference to the faster OCSP route and only demand one source to succeed) is a good choice for most typical validation environments. The "crcAll*" modes are much stricter, and may be used in scenarios where bulletproof validity information is essential.

NOTE: If no CRL or OCSP endpoints are provided by the CA, the revocation check will be considered successful. This is because the CA chose not to supply revocation information for its certificates, meaning they are considered irrevocable.

NOTE: Within each of the above settings, if any retrieved CRL or OCSP response indicates that the certificate has been revoked, the revocation check fails.

Signatures Property (XAdESSigner Component)

Provides details of all signatures found in the XML document.

Syntax

C# VB.NET

```
public XAdESSignatureList Signatures { get; }
```

Remarks

Use this property to get the details of all the signatures identified in the XML document.

This property is read-only and not available at design time.

Please refer to the XAdESSignature type for a complete list of fields.

SigningCertificate Property (XAdESSigner Component)

The certificate to be used for signing.

Syntax

C# VB.NET

```
public Certificate SigningCertificate { get; set; }
```

Remarks

Use this property to specify the certificate that shall be used for signing the data. Note that this certificate should have a private key associated with it. Use SigningChain to supply the rest of the certificate chain for inclusion into the signature.

This property is not available at design time.

Please refer to the Certificate type for a complete list of fields.

SigningChain Property (XAdESSigner Component)

The signing certificate chain.

Syntax

C# VB.NET

```
public CertificateList SigningChain { get; }
```

Remarks

Use this property to provide the chain for the signing certificate. Use the SigningCertificate property, if it is available, to provide the signing certificate itself.

This property is not available at design time.

Please refer to the Certificate type for a complete list of fields.

SocketSettings Property (XAdESSigner Component)

Manages network connection settings.

Syntax

C# VB.NET

```
public SocketSettings SocketSettings { get; }
```

Remarks

Use this property to tune up network connection parameters.

This property is read-only.

Please refer to the SocketSettings type for a complete list of fields.

Timestamps Property (XAdESSigner Component)

Contains a collection of timestamps for the processed document.

Syntax

C# VB.NET

```
public TimestampInfoList Timestamps { get; }
```

Remarks

Use this property to access the timestamps included in the processed document.

This property is read-only and not available at design time.

Please refer to the TimestampInfo type for a complete list of fields.

TimestampServer Property (XAdESSigner Component)

The address of the timestamping server.

Syntax

C# VB.NET

```
public string TimestampServer { get; set; }
```

Default Value

""

Remarks

Use this property to provide the address of the Time Stamping Authority (TSA) server to be used for timestamping the signature.

SecureBlackbox supports RFC3161-compliant timestamping servers, available via HTTP or HTTPS.

If your timestamping service enforces credential-based user authentication (basic or digest), you can provide the credentials in the same URL:

http://user:password@timestamp.server.com/TsaService

For TSAs using certificate-based TLS authentication, provide the client certificate via the TLSClientChain property.

If this property is left empty, no timestamp will be added to the signature.

Starting from summer 2021 update (Vol. 2), the *virtual* timestamping service is supported, which allows you to intervene in the timestamping routine and provide your own handling for the TSA exchange. This may be handy if the service that you are requesting timestamps from uses a non-standard TSP protocol or requires special authentication option.

To employ the virtual service, assign an URI of the following format to this property:

virtual://localhost?
hashonly=true&includecerts=true&reqpolicy=1.2.3.4.5&halg=SHA256&ignorenonce=

Subscribe to Notification event to get notified about the virtualized timestamping event. The EventID of the timestamping event is TimestampRequest. Inside the event handler, read the base16-encoded request from the EventParam parameter and forward it to the timestamping authority. Upon receiving the response, pass it back to the component, encoded in base16, via the TimestampResponse config property:

component.Config("TimestampResponse=308208ab...");

Note that all the exchange with your custom TSA should take place within the same invocation of the Notification event.

The *hashonly* parameter of the virtual URI tells the component to only return the timestamp message imprint via the EventParam parameter. If set to false, EventParam will contain the complete RFC3161 timestamping request.

The *includecerts* parameter specifies that the requestCertificates parameter of the timestamping request should be set to true.

The *reqpolicy* parameter lets you specify the request policy, and the *halg* parameter specifies the hash algorithm to use for timestamping.

The *ignorenonce* parameter allows you to switch off client nonce verification to enable compatibility with TSA services that do not support nonce mirroring.

All the parameters are optional.

TLSClientChain Property (XAdESSigner Component)

The TLS client certificate chain.

Syntax

C# VB.NET

```
public CertificateList TLSClientChain { get; }
```

Remarks

Assign a certificate chain to this property to enable TLS client authentication in the component. Note that the client's end-entity certificate should have a private key associated with it.

Use the CertificateStorage or CertificateManager components to import the certificate from a file, system store, or PKCS11 device.

This property is not available at design time.

Please refer to the Certificate type for a complete list of fields.

TLSServerChain Property (XAdESSigner Component)

The TLS server's certificate chain.

Syntax

C# VB.NET

```
public CertificateList TLSServerChain { get; }
```

Remarks

Use this property to access the certificate chain sent by the TLS server. This property is ready to read when the TLSCertValidate event is fired by the client component.

This property is read-only and not available at design time.

Please refer to the Certificate type for a complete list of fields.

TLSSettings Property (XAdESSigner Component)

Manages TLS layer settings.

Syntax

C# VB.NET

```
public TLSSettings TLSSettings { get; }
```

Remarks

Use this property to tune up the TLS layer parameters.

This property is read-only.

Please refer to the TLSSettings type for a complete list of fields.

TrustedCertificates Property (XAdESSigner Component)

A list of trusted certificates for chain validation.

Syntax

C# VB.NET

```
public CertificateList TrustedCertificates { get; }
```

Remarks

Use this property to supply a list of trusted certificates that might be needed for chain validation. An example of a scenario where you might want to do that is when root CA certificates are absent from the standard system locations (or when there are no standard system locations), and therefore should be supplied to the component manually.

The purpose of this certificate collection is largely the same as that of the Windows Trusted Root Certification Authorities system store.

Use this property with extreme care as it directly affects chain verifiability; a wrong certificate added to the trusted list may result in bad chains being accepted, and forfeited signatures being recognized as genuine. Only add certificates that originate from the parties that you know and trust.

This property is not available at design time.

Please refer to the Certificate type for a complete list of fields.

ValidationMoment Property (XAdESSigner Component)

The time point at which signature validity is to be established.

Syntax

C# VB.NET

```
public string ValidationMoment { get; set; }
```

Default Value

""

Remarks

Use this property to specify the moment in time at which signature validity should be established. The time is in UTC. Leave the setting empty to stick to the default moment (either the signature creation time or the current time).

The validity of the same signature may differ depending on the time point chosen due to temporal changes in chain validities, revocation statuses, and timestamp times.

AddDataReference Method (XAdESSigner Component)

Creates a new XML reference to the specified data.

Syntax

C# VB.NET

```
public int AddDataReference(string dataURI, byte[] data);
```

Remarks

Use this method to add a reference to the custom data. Pass the reference's URI via *DataURI* parameter.

This method uses HashAlgorithm property to specify the hash algorithm of the reference.

The method returns the index of the new reference entry in the References collection.

AddKnownNamespace Method (XAdESSigner Component)

Adds known prefix and correspondent namespace URI.

Syntax

C# VB.NET

```
public void AddKnownNamespace(string prefix, string URI);
```

Remarks

Use this method to add a known prefix and namespace URI that are used in XPath expression within XMLElement/XMLNode property, and within TargetXMLElement and XPathPrefixList properties of the references.

AddReference Method (XAdESSigner Component)

Creates a new XML reference to the specified XML element.

Syntax

C# VB.NET

```
public int AddReference(string targetXmlElement, string customId, bool autoGenerateId);
```

Remarks

Use this method to add a reference to a particular XML element.

The reference's *URI* is set basing on the ID of the XML element. If the XML element doesn't have an ID then a *CustomId* value will be used. If *CustomId* is empty and *AutoGenerateId* is set, the ID will be generated automatically. An exception will be thrown otherwise.

This method uses CanonicalizationMethod and HashAlgorithm properties to specify the canonicalization method and hash algorithm of the reference.

The method returns the index of the new reference entry in the References collection.

Close Method (XAdESSigner Component)

Closes an opened document.

Syntax

C# VB.NET

```
public void Close(bool saveChanges);
```

Remarks

Use this method to close a previously opened document. Set *SaveChanges* to true to apply any changes made.

Config Method (XAdESSigner Component)

Sets or retrieves a configuration setting.

Syntax

C# VB.NET

```
public string Config(string configurationString);
```

Remarks

Config is a generic method available in every component. It is used to set and retrieve configuration settings for the component.

These settings are similar in functionality to properties, but they are rarely used. In order to avoid "polluting" the property namespace of the component, access to these *internal properties* is provided through the Config method.

To set a configuration setting named *PROPERTY*, you must call *Config("PROPERTY=VALUE")*, where *VALUE* is the value of the setting expressed as a string. For boolean values, use the strings "True", "False", "0", "1", "Yes", or "No" (case does not matter).

To read (query) the value of a configuration setting, you must call *Config("PROPERTY")*. The value will be returned as a string.

DoAction Method (XAdESSigner Component)

Performs an additional action.

Syntax

C# VB.NET

```
public string DoAction(string actionID, string actionParams);
```

Remarks

DoAction is a generic method available in every component. It is used to perform an additional action introduced after the product major release. The list of actions is not fixed, and may be flexibly extended over time.

The unique identifier (case insensitive) of the action is provided in the *ActionID* parameter.

ActionParams contains the value of a single parameter, or a list of multiple parameters for the action in the form of *PARAM1=VALUE1;PARAM2=VALUE2;...*.

Common ActionIDs:

Action	*Parameters*	*Returned value*	*Description*
ResetTrustedListCache	none	none	Clears the cached list of trusted lists.
ResetCertificateCache	none	none	Clears the cached certificates.
ResetCRLCache	none	none	Clears the cached CRLs.
ResetOCSPResponseCache	none	none	Clears the cached OCSP responses.

ExtractAsyncData Method (XAdESSigner Component)

Extracts user data from the DC signing service response.

Syntax

C# VB.NET

```
public string ExtractAsyncData(string asyncReply);
```

Remarks

Call this method before finalizing the asynchronous signing process to extract the data passed to the ExternalCrypto.Data property on the pre-signing stage.

The Data parameter can be used to pass some state or document identifier along with the signing request from the pre-signing to the completion async stage.

GetInnerXML Method (XAdESSigner Component)

Get the inner XML content of the selected XML element.

Syntax

C# VB.NET

```
public string GetInnerXML(string XPath);
```

Remarks

Call this method to get the inner XML content of the selected XML element.

GetOuterXML Method (XAdESSigner Component)

Get the outer XML content of the selected XML element.

Syntax

C# VB.NET

```
public string GetOuterXML(string XPath);
```

Remarks

Call this method to get the outer XML content of the selected XML element.

GetTextContent Method (XAdESSigner Component)

Get the text content of the selected XML element.

Syntax

C# VB.NET

```
public string GetTextContent(string XPath);
```

Remarks

Call this method to get the text content of the selected XML element.

Open Method (XAdESSigner Component)

Opens a document for signing or updating.

Syntax

C# VB.NET

```
public void Open();
```

Remarks

Use this method to open a document for signing or updating. When finished, call Close to complete or discard the operation.

Reset Method (XAdESSigner Component)

Resets the component settings.

Syntax

C# VB.NET

```
public void Reset();
```

Remarks

Reset is a generic method available in every component.

Revalidate Method (XAdESSigner Component)

Revalidates a signature in accordance with current settings.

Syntax

C# VB.NET

```
public void Revalidate(string sigLabel, bool detached);
```

Remarks

Use this method to re-validate a signature in the opened XML document.

SelectInfo Method (XAdESSigner Component)

Select signature information for a specific entity.

Syntax

C# VB.NET

```
public void SelectInfo(string entityLabel, int infoType, bool clearSelection);
```

Remarks

Use this method to select (or filter) signature-related information for a specific signature element.

Provide the unique label of the entity that you are interested in via the *EntityLabel* parameter. Use one of the following filters, or their combination, to specify what information you are interested in:

sitEntity	`1`	Select the current entity
sitParentEntity	`2`	Select the parent entity of the current entity
sitTimestamps	`4`	Select all timestamps covering the current entity
sitSignatures	`8`	Select all signatures covering the current entity
sitSigningChain	`16`	Select the signing chain of the current entity
sitEmbeddedCertificates	`256`	Select all certificates embedded in the current entity
sitEmbeddedCRLs	`512`	Select all CRLs embedded in the current entity
sitEmbeddedOCSPs	`1024`	Select all OCSP responses embedded in the current entity
sitEmbeddedRevInfo	`1792`	Select the whole pack of embedded revocation information (certificates, CRLs and OCSPs)
sitUsedCertificates	`4096`	Select all the certificates used to validate this entity's chain
sitUsedCRLs	`8192`	Select all the CRLs used to validate this entity's chain
sitUsedOCSPs	`16384`	Select all the OCSP responses used to validate this entity's chain
sitUsedRevInfo	`28672`	Select the whole pack of revocation information used to validate this entity's chain (certificates, CRLs, OCSP responses)
sitAttributes	`65536`	Select this entity's CMS attributes
sitReferences	`131072`	Select this entity's XML references

sitSignedParts	262144	Select this entity's signed parts

Following the call, the relevant pieces of information will be copied to the respective component properties (Certificates, CRLs, OCSPs). Note that you can accumulate information in the properties by making repeated calls to SelectInfo and keeping *ClearSelection* set to false.

This method is useful if you would like to read/display detailed information about a particular signature or timestamp.

SetInnerXML Method (XAdESSigner Component)

Set the inner XML content of the selected XML element.

Syntax

C# VB.NET

```
public void SetInnerXML(string XPath, string value);
```

Remarks

Call this method to set the inner XML content of the selected XML element.

SetTextContent Method (XAdESSigner Component)

Set the text content of the selected XML element.

Syntax

C# VB.NET

```
public void SetTextContent(string XPath, string value);
```

Remarks

Call this method to set the text content of the selected XML element.

Sign Method (XAdESSigner Component)

Signs an XML document.

Syntax

C# VB.NET

```
public void Sign();
```

Remarks

Call this method to generate a signature over an XML document.

SignAsyncBegin Method (XAdESSigner Component)

Initiates the asynchronous signing operation.

Syntax

C# VB.NET

```
public string SignAsyncBegin();
```

Remarks

When using the DC framework, call this method to initiate the asynchronous signing process. Upon completion, a pre-signed copy of the document will be saved in OutputFile (or OutputStream). Keep the pre-signed copy somewhere local, and pass the returned string ('the request state') to the DC processor for handling.

Upon receiving the response state from the DC processor, assign the path to the pre-signed copy to InputFile (or InputStream), and call SignAsyncEnd to finalize the signing.

Note that depending on the signing method and DC configuration used, you may still need to provide the public part of the signing certificate via the SigningCertificate property.

Use the *ExternalCrypto.AsyncDocumentID* property to supply a unique document ID to include in the request. This is helpful when creating batches of multiple async requests, as it allows you to pass the whole response batch to SignAsyncEnd and expect it to recover the correct response from the batch automatically.

AsyncState is a message of the distributed cryptography (DC) protocol. The DC protocol is based on the exchange of async states between a DC client (an application that wants to sign a PDF, XML, or Office document) and a DC server (an application that controls access to the private key). An async state can carry one or more signing requests, comprised of document hashes, or one or more signatures produced over those hashes.

In a typical scenario you get a client-side async state from the SignAsyncBegin method. This state contains document hashes to be signed on the DC server side. You then send the async state to the DC server (often represented by the DCAuth component), which processes it and produces a matching signature state. The async state produced by the server is then passed to the SignAsyncEnd method.

SignAsyncEnd Method (XAdESSigner Component)

Completes the asynchronous signing operation.

Syntax

C# VB.NET

```
public void SignAsyncEnd(string asyncReply);
```

Remarks

When using the DC framework, call this method upon receiving the response state from the DC processor to complete the asynchronous signing process.

Before calling this method, assign the path to the pre-signed copy of the document obtained from the prior SignAsyncBegin call to InputFile (or InputStream). The method will embed the signature into the pre-signed document, and save the complete signed document to OutputFile (or OutputStream).

Note that depending on the signing method and DC configuration used, you may still need to provide the public part of the signing certificate via the SigningCertificate property.

Use the *ExternalCrypto.AsyncDocumentID* parameter to pass a specific document ID if using batched AsyncReply. If used, it should match the value provided on the pre-signing (SignAsyncBegin) stage.

AsyncState is a message of the distributed cryptography (DC) protocol. The DC protocol is based on the exchange of async states between a DC client (an application that wants to sign a PDF, XML, or Office document) and a DC server (an application that controls access to the private key). An async state can carry one or more signing requests, comprised of document hashes, or one or more signatures produced over those hashes.

In a typical scenario you get a client-side async state from the SignAsyncBegin method. This state contains document hashes to be signed on the DC server side. You then send the async state to the DC server (often represented by the DCAuth component), which processes it and produces a matching signature state. The async state produced by the server is then passed to the SignAsyncEnd method.

SignExternal Method (XAdESSigner Component)

Signs the document using an external signing facility.

Syntax

C# VB.NET

```
public void SignExternal();
```

Remarks

Call this method to delegate the low-level signing operation to an external, remote, or custom signing engine. This method is useful if the signature has to be made by a device accessible through a custom or non-standard signing interface.

When all preparations are done and hash is computed, the component fires ExternalSign event which allows to pass the hash value for signing.

Timestamp Method (XAdESSigner Component)

Use this method to add an timestamp.

Syntax

C# VB.NET

```
public void Timestamp(string sigLabel, int timestampType);
```

Remarks

Call this method to timestamp the signature. Use the *TimestampServer* property to provide the address of the TSA (Time Stamping Authority) server which should be used for timestamping. This method could be called separately or in SignatureValidated event handler after successful signature validation. Use the *TimestampType* parameter to specify the type of timestamp to create

Supported timestamp types:

tstSignature	12	Signature timestamp
tstRefsOnly	13	RefsOnly timestamp
tstSigAndRefs	14	SigAndRefs timestamp
tstArchive	7	Archive timestamp

Upgrade Method (XAdESSigner Component)

Upgrades existing XAdES signature to a new form.

Syntax

C# VB.NET

```
public void Upgrade(string sigLabel, int upgradeKind);
```

Remarks

XAdES standard defines a number of different 'forms' of signatures which can be used for different purposes. Use this method to upgrade XAdES signature to a new form or level specified by *UpgradeKind*. Signatures can normally be upgraded from less sophisticated levels (BES, EPES) to more sophisticated (T, C, X, X-L, A).

The supported levels and forms are:

aslUnknown	`0`	Unknown signature level
aslGeneric	`1`	Generic (this value applicable to XAdES signature only and corresponds to XML-DSIG signature)
aslBaselineB	`2`	Baseline B (B-B, basic)
aslBaselineT	`3`	Baseline T (B-T, timestamped)
aslBaselineLT	`4`	Baseline LT (B-LT, long-term)
aslBaselineLTA	`5`	Baseline LTA (B-LTA, long-term with archived timestamp)
aslBES	`6`	BES (Basic Electronic Signature)
aslEPES	`7`	EPES (Electronic Signature with an Explicit Policy)
aslT	`8`	T (Timestamped)
aslC	`9`	C (T with revocation references)
aslX	`10`	X (C with SigAndRefs timestamp or RefsOnly timestamp) (this value applicable to XAdES signature only)
aslXType1	`11`	X Type 1 (C with an ES-C timestamp) (this value applicable to CAdES signature only)
aslXType2	`12`	X Type 2 (C with a CertsAndCRLs timestamp) (this value applicable to CAdES signature only)
aslXL	`13`	X-L (X with revocation values) (this value applicable to XAdES signature only)
aslXLType1	`14`	X-L Type 1 (C with revocation values and an ES-C timestamp) (this value applicable to CAdES signature only)

aslXLType2	15	X-L Type 2 (C with revocation values and a CertsAndCRLs timestamp) (this value applicable to CAdES signature only)
aslA	16	A (archived)
aslExtendedBES	17	Extended BES
aslExtendedEPES	18	Extended EPES
aslExtendedT	19	Extended T
aslExtendedC	20	Extended C
aslExtendedX	21	Extended X (this value applicable to XAdES signature only)
aslExtendedXType1	22	Extended X (type 1) (this value applicable to CAdES signature only)
aslExtendedXType2	23	Extended X (type 2) (this value applicable to CAdES signature only)
aslExtendedXLong	24	Extended X-Long (this value applicable to XAdES signature only)
aslExtendedXL	25	Extended X-L (this value applicable to XAdES signature only)
aslExtendedXLType1	26	Extended XL (type 1) (this value applicable to CAdES signature only)
aslExtendedXLType2	27	Extended XL (type 2) (this value applicable to CAdES signature only)
aslExtendedA	28	Extended A

The supported additional upgrade kinds are:

xukAddValidationDataRefs	256	Add signature validation references
xukAddValidationDataValues	512	Add signature validation values

xukAddTimestampValidationData	1024	Add timestamp validation data
xukAddAnyValidationData	2048	Add AnyValidationData

ChainElementDownload Event (XAdESSigner Component)

Fires when there is a need to download a chain element from an online source.

Syntax

C# VB.NET

```
public event OnChainElementDownloadHandler OnChainElementDownload;

public delegate void OnChainElementDownloadHandler(object sender, XAdESSignerChainElement

public class XAdESSignerChainElementDownloadEventArgs : EventArgs {
  public int Kind { get; }
  public string CertRDN { get; }
  public string CACertRDN { get; }
  public string Location { get; }
  public int Action { get; set; }
}
```

Remarks

Subscribe to this event to be notified about validation element retrievals. Use the *Action* parameter to suppress the download if required.

veaAuto	0	Handle the action automatically (the default behaviour)
veaContinue	1	Accept the request implied by the event (accept the certificate, allow the object retrieval)
veaReject	2	Reject the request implied by the event (reject the certificate, disallow the object retrieval)

veaAcceptNow	3	Accept the validated certificate immediately
veaAbortNow	4	Abort the validation, reject the certificate

cekUnknown	0	Unknown or unsupported element type
cekCertificate	1	An X.509 certificate
cekCRL	2	A CRL
cekOCSP	3	An OCSP response

ChainElementNeeded Event (XAdESSigner Component)

Fires when an element required to validate the chain was not located.

Syntax

C# VB.NET

```
public event OnChainElementNeededHandler OnChainElementNeeded;

public delegate void OnChainElementNeededHandler(object sender, XAdESSignerChainElementNe

public class XAdESSignerChainElementNeededEventArgs : EventArgs {
  public int Kind { get; }
  public string CertRDN { get; }
  public string CACertRDN { get; }
}
```

Remarks

Subscribe to this event to be notified about missing validation elements. Use the KnownCRLs, KnownCertificates, and KnownOCSPs properties in the event handler to provide the missing piece.

cekUnknown	0	Unknown or unsupported element type
cekCertificate	1	An X.509 certificate
cekCRL	2	A CRL
cekOCSP	3	An OCSP response

ChainElementStore Event (XAdESSigner Component)

This event is fired when a chain element (certificate, CRL, or OCSP response) should be stored along with a signature.

Syntax

C# VB.NET

```
public event OnChainElementStoreHandler OnChainElementStore;

public delegate void OnChainElementStoreHandler(object sender, XAdESSignerChainElementSto

public class XAdESSignerChainElementStoreEventArgs : EventArgs {
  public int Kind { get; }
  public byte[] Body { get; }
  public string URI { get; set; }
}
```

Remarks

This event could occur if you are verifying XAdES-C form or higher. The *Body* parameter contains the element in binary form that should be stored along with a signature. Use the *URI* parameter to provide an URI of the stored element.

cekUnknown	0	Unknown or unsupported element type

cekCertificate	1	An X.509 certificate
cekCRL	2	A CRL
cekOCSP	3	An OCSP response

ChainValidated Event (XAdESSigner Component)

Reports the completion of a certificate chain validation.

Syntax

C# VB.NET

```
public event OnChainValidatedHandler OnChainValidated;

public delegate void OnChainValidatedHandler(object sender, XAdESSignerChainValidatedEver

public class XAdESSignerChainValidatedEventArgs : EventArgs {
  public int Index { get; }
  public string EntityLabel { get; }
  public string SubjectRDN { get; }
  public int ValidationResult { get; }
  public int ValidationDetails { get; }
  public bool Cancel { get; set; }
}
```

Remarks

This event is fired when a certificate chain validation routine completes. *SubjectRDN* identifies the owner of the validated certificate.

ValidationResult set to 0 (zero) indicates successful chain validation.

cvtValid	0	The chain is valid

cvtValidButUntrusted	`1`	The chain is valid, but the root certificate is not trusted
cvtInvalid	`2`	The chain is not valid (some of certificates are revoked, expired, or contain an invalid signature)
cvtCantBeEstablished	`3`	The validity of the chain cannot be established because of missing or unavailable validation information (certificates, CRLs, or OCSP responses)

Any other value reports a failure, and *ValidationDetails* provides more details on its reasons.

cvrBadData	`0x0001`	One or more certificates in the validation path are malformed
cvrRevoked	`0x0002`	One or more certificates are revoked
cvrNotYetValid	`0x0004`	One or more certificates are not yet valid
cvrExpired	`0x0008`	One or more certificates are expired
cvrInvalidSignature	`0x0010`	A certificate contains a non-valid digital signature
cvrUnknownCA	`0x0020`	A CA certificate for one or more certificates has not been found (chain incomplete)
cvrCAUnauthorized	`0x0040`	One of the CA certificates are not authorized to act as CA
cvrCRLNotVerified	`0x0080`	One or more CRLs could not be verified
cvrOCSPNotVerified	`0x0100`	One or more OCSP responses could not be verified
cvrIdentityMismatch	`0x0200`	The identity protected by the certificate (a TLS endpoint or an e-mail addressee) does not match what is recorded in the certificate
cvrNoKeyUsage	`0x0400`	A mandatory key usage is not enabled in one of the chain certificates

cvrBlocked	`0x0800`	One or more certificates are blocked
cvrFailure	`0x1000`	General validation failure
cvrChainLoop	`0x2000`	Chain loop: one of the CA certificates recursively signs itself
cvrWeakAlgorithm	`0x4000`	A weak algorithm is used in one of certificates or revocation elements
cvrUserEnforced	`0x8000`	The chain was considered invalid following intervention from a user code

ChainValidationProgress Event (XAdESSigner Component)

This event is fired multiple times during chain validation to report various stages of the validation procedure.

Syntax

C# VB.NET

```
public event OnChainValidationProgressHandler OnChainValidationProgress;

public delegate void OnChainValidationProgressHandler(object sender, XAdESSignerChainVali

public class XAdESSignerChainValidationProgressEventArgs : EventArgs {
  public string EventKind { get; }
  public string CertRDN { get; }
  public string CACertRDN { get; }
  public int Action { get; set; }
}
```

Remarks

Subscribe to this event to be notified about chain validation progress. Use the *Action* parameter to alter the validation flow.

The *EventKind* parameter reports the nature of the event being reported. The *CertRDN* and *CACertRDN* parameters report the distinguished names of the certificates that are relevant for the event invocation (one or both can be empty, depending on *EventKind*). Use the *Action* parameter to adjust the validation flow.

veaAuto	`0`	Handle the action automatically (the default behaviour)
veaContinue	`1`	Accept the request implied by the event (accept the certificate, allow the object retrieval)
veaReject	`2`	Reject the request implied by the event (reject the certificate, disallow the object retrieval)
veaAcceptNow	`3`	Accept the validated certificate immediately
veaAbortNow	`4`	Abort the validation, reject the certificate

DocumentLoaded Event (XAdESSigner Component)

This event is fired when the document has been loaded into memory.

Syntax

C# VB.NET

```
public event OnDocumentLoadedHandler OnDocumentLoaded;

public delegate void OnDocumentLoadedHandler(object sender, XAdESSignerDocumentLoadedEver

public class XAdESSignerDocumentLoadedEventArgs : EventArgs {
  public bool Cancel { get; set; }
}
```

Remarks

The handler for this event is a good place to check document properties, which may be useful when preparing the signature, for example, the document format.

Set *Cancel* to true to terminate document processing on this stage.

Error Event (XAdESSigner Component)

Information about errors during signing.

Syntax

C# VB.NET

```
public event OnErrorHandler OnError;

public delegate void OnErrorHandler(object sender, XAdESSignerErrorEventArgs e);

public class XAdESSignerErrorEventArgs : EventArgs {
  public int ErrorCode { get; }
  public string Description { get; }
}
```

Remarks

The event is fired in case of exceptional conditions during signing.

ErrorCode contains an error code and *Description* contains a textual description of the error. For a list of valid error codes and their descriptions, please refer to XML.

ExternalSign Event (XAdESSigner Component)

Handles remote or external signing initiated by the SignExternal method or other source.

Syntax

C# VB.NET

```
public event OnExternalSignHandler OnExternalSign;
```

```
public delegate void OnExternalSignHandler(object sender, XAdESSignerExternalSignEventArg

public class XAdESSignerExternalSignEventArgs : EventArgs {
  public string OperationId { get; }
  public string HashAlgorithm { get; }
  public string Pars { get; }
  public string Data { get; }
  public string SignedData { get; set; }
}
```

Remarks

Assign a handler to this event if you need to delegate a low-level signing operation to an external, remote, or custom signing engine. Depending on the settings, the handler will receive a hashed or unhashed value to be signed.

The event handler must pass the value of *Data* to the signer, obtain the signature, and pass it back to the component via the *SignedData* parameter.

OperationId provides a comment about the operation and its origin. It depends on the exact component being used, and may be empty. *HashAlgorithm* specifies the hash algorithm being used for the operation, and *Pars* contains algorithm-dependent parameters.

The component uses base16 (hex) encoding for the *Data*, *SignedData*, and *Pars* parameters. If your signing engine uses a different input and output encoding, you may need to decode and/or encode the data before and/or after the signing.

A sample MD5 hash encoded in base16: a0dee2a0382afbb09120ffa7ccd8a152 - lower case base16
A0DEE2A0382AFBB09120FFA7CCD8A152 - upper case base16

A sample event handler that uses the .NET RSACryptoServiceProvider class may look like the following:

Copy

```
signer.OnExternalSign += (s, e) =>
{
        var cert = new X509Certificate2("cert.pfx", "", X509KeyStorageFlags.Exportable);
        var key = (RSACryptoServiceProvider)cert.PrivateKey;

        var dataToSign = e.Data.FromBase16String();
        var signedData = key.SignHash(dataToSign, "2.16.840.1.101.3.4.2.1");
        e.SignedData = signedData.ToBase16String();
};
```

FormatElement Event (XAdESSigner Component)

Reports the XML element that is currently being processed.

Syntax

C# VB.NET

```
public event OnFormatElementHandler OnFormatElement;

public delegate void OnFormatElementHandler(object sender, XAdESSignerFormatElementEventA

public class XAdESSignerFormatElementEventArgs : EventArgs {
  public string StartTagWhitespace { get; set; }
  public string EndTagWhitespace { get; set; }
  public int Level { get; }
  public string Path { get; }
  public bool HasChildElements { get; }
}
```

Remarks

Path and *Level* specify the path to the XML element being processed and its nesting level, respectively.

HasChildElements specify if processed XML element has child elements.

Among other purposes, this event may be used to add whitespace formatting before or after a particular element in the signature.

FormatText Event (XAdESSigner Component)

Reports XML text that is currently being processed.

Syntax

C# VB.NET

```
public event OnFormatTextHandler OnFormatText;

public delegate void OnFormatTextHandler(object sender, XAdESSignerFormatTextEventArgs e)

public class XAdESSignerFormatTextEventArgs : EventArgs {
  public string Text { get; set; }
```

```
  public int TextType { get; }
  public int Level { get; }
  public string Path { get; }
}
```

Remarks

TextType parameter specifies the type of the XML text (normal or Base64-encoded) that is stored in the element; *Path* and *Level* specify the path to the XML element and its nesting level.

Among other purposes, this event may be used to add whitespace formatting before or after a particular element in the signature.

Notification Event (XAdESSigner Component)

This event notifies the application about an underlying control flow event.

Syntax

C# VB.NET

```
public event OnNotificationHandler OnNotification;

public delegate void OnNotificationHandler(object sender, XAdESSignerNotificationEventArg

public class XAdESSignerNotificationEventArgs : EventArgs {
  public string EventID { get; }
  public string EventParam { get; }
}
```

Remarks

The component fires this event to let the application know about some event, occurrence, or milestone in the component. For example, it may fire to report completion of the document processing. The list of events being reported is not fixed, and may be flexibly extended over time.

The unique identifier of the event is provided in the *EventID* parameter. *EventParam* contains any parameters accompanying the occurrence. Depending on the type of the component, the exact action it is performing, or the document being processed, one or both may be omitted.

This component can fire this event with the following *EventID* values:

BeforeTimestamp	This event is fired before a timestamp is requested from the timestamping authority. Use the event handler to modify TSA and HTTP settings.
TimestampError	This event is only fired if the component failed to obtain a timestamp from the timestamping authority. The *EventParam* parameter contains extended error info.
TimestampRequest	A timestamp is requested from the custom timestamping authority. This event is only fired if TimestampServer was set to a *virtual://* URI. The *EventParam* parameter contains the TSP request (or the plain hash, depending on the value provided to TimestampServer), in base16, that needs to be sent to the TSA. Use the event handler to send the request to the TSA. Upon receiving the response, assign it, in base16, to the *TimestampResponse* configuration property.

ReferenceValidated Event (XAdESSigner Component)

Marks the end of a reference validation.

Syntax

C# VB.NET

```
public event OnReferenceValidatedHandler OnReferenceValidated;

public delegate void OnReferenceValidatedHandler(object sender, XAdESSignerReferenceValid

public class XAdESSignerReferenceValidatedEventArgs : EventArgs {
  public int ReferenceIndex { get; }
  public string ID { get; }
  public string URI { get; }
  public string RefType { get; }
  public bool DigestValid { get; }
}
```

Remarks

The component fires this event to report completion of a reference validation. A reference is a building block of a signature as it binds signature coverage to a particular piece of the document.

ResolveReference Event (XAdESSigner Component)

Asks the application to resolve a reference.

Syntax

C# VB.NET

```
public event OnResolveReferenceHandler OnResolveReference;

public delegate void OnResolveReferenceHandler(object sender, XAdESSignerResolveReference

public class XAdESSignerResolveReferenceEventArgs : EventArgs {
  public int ReferenceIndex { get; }
  public string URI { get; }
}
```

Remarks

This event is fired when the control could not automatically resolve a reference and requires custom treatment.

URI contains a reference to the data.

ReferenceIndex specifies the index of the reference to process.

Based on the reference's URI the event handler should set either TargetXMLElement or TargetData property of the reference.

SignatureFound Event (XAdESSigner Component)

Signifies the start of signature validation.

Syntax

C# VB.NET

```
public event OnSignatureFoundHandler OnSignatureFound;

public delegate void OnSignatureFoundHandler(object sender, XAdESSignerSignatureFoundEver

public class XAdESSignerSignatureFoundEventArgs : EventArgs {
  public int Index { get; }
  public string EntityLabel { get; }
  public string IssuerRDN { get; }
  public byte[] SerialNumber { get; }
  public byte[] SubjectKeyID { get; }
  public bool CertFound { get; }
  public bool ValidateSignature { get; set; }
  public bool ValidateChain { get; set; }
}
```

Remarks

This event tells the application that signature validation is about to start, and provides the details about the signer's certificate via its *IssuerRDN*, *SerialNumber*, and *SubjectKeyID* parameters. It fires for every signature located in the verified document or message.

The *CertFound* parameter is set to True if the component has found the needed certificate in one of the known locations, and to False otherwise, in which case you must provide it manually via the KnownCertificates property.

Signature validation consists of two independent stages: cryptographic signature validation and chain validation. Separate validation results are reported for each, with the SignatureValidationResult and ChainValidationResult properties respectively.

Use the *ValidateSignature* and *ValidateChain* parameters to tell the verifier which stages to include in the validation.

SignatureValidated Event (XAdESSigner Component)

Marks the completion of the signature validation routine.

Syntax

C# VB.NET

```
public event OnSignatureValidatedHandler OnSignatureValidated;

public delegate void OnSignatureValidatedHandler(object sender, XAdESSignerSignatureValid

public class XAdESSignerSignatureValidatedEventArgs : EventArgs {
  public int Index { get; }
  public string EntityLabel { get; }
  public string IssuerRDN { get; }
  public byte[] SerialNumber { get; }
  public byte[] SubjectKeyID { get; }
  public int ValidationResult { get; }
  public bool Cancel { get; set; }
}
```

Remarks

This event is fired upon the completion of the signature validation routine, and reports the respective validation result.

Use the *IssuerRDN*, *SerialNumber*, and/or *SubjectKeyID* parameters to identify the signing certificate.

ValidationResult is set to 0 if the validation has been successful, or to a non-zero value in case of a validation failure.

svtValid	0	The signature is valid
svtUnknown	1	Signature validity is unknown
svtCorrupted	2	The signature is corrupted
svtSignerNotFound	3	Failed to acquire the signing certificate. The signature cannot be validated.
svtFailure	4	General failure
svtReferenceCorrupted	5	Reference corrupted (XML-based signatures only)

TimestampFound Event (XAdESSigner Component)

Signifies the start of a timestamp validation routine.

Syntax

C# VB.NET

```
public event OnTimestampFoundHandler OnTimestampFound;

public delegate void OnTimestampFoundHandler(object sender, XAdESSignerTimestampFoundEver

public class XAdESSignerTimestampFoundEventArgs : EventArgs {
  public int Index { get; }
  public string EntityLabel { get; }
  public string IssuerRDN { get; }
  public byte[] SerialNumber { get; }
  public byte[] SubjectKeyID { get; }
  public bool CertFound { get; }
  public bool ValidateTimestamp { get; set; }
  public bool ValidateChain { get; set; }
}
```

Remarks

This event fires for every timestamp identified during signature processing, and reports the details about the signer's certificate via its *IssuerRDN*, *SerialNumber*, and *SubjectKeyID* parameters.

The *CertFound* parameter is set to True if the component has found the needed certificate in one of the known locations, and to False otherwise, in which case you must provide it manually via the KnownCertificates property.

Just like with signature validation, timestamp validation consists of two independent stages: cryptographic signature validation and chain validation. Separate validation results are reported for each, with the ValidationResult and ChainValidationResult properties respectively.

Use the *ValidateSignature* and *ValidateChain* parameters to tell the verifier which stages to include in the validation.

TimestampRequest Event (XAdESSigner Component)

Fires when the component is ready to request a timestamp from an external TSA.

Syntax

C# VB.NET

```
public event OnTimestampRequestHandler OnTimestampRequest;

public delegate void OnTimestampRequestHandler(object sender, XAdESSignerTimestampRequest

public class XAdESSignerTimestampRequestEventArgs : EventArgs {
  public string TSA { get; }
  public string TimestampRequest { get; }
  public string TimestampResponse { get; set; }
  public bool SuppressDefault { get; set; }
}
```

Remarks

Subscribe to this event to intercept timestamp requests. You can use it to override timestamping requests and perform them in your code.

The *TSA* parameter indicates the timestamping service being used. It matches the value passed to the TimestampServer property. Set the *SuppressDefault* parameter to true if you would like to stop the built-in TSA request from going ahead. The built-in TSA request is also not performed if the returned *TimestampResponse* parameter is not empty.

TimestampValidated Event (XAdESSigner Component)

Reports the completion of the timestamp validation routine.

Syntax

C# VB.NET

```
public event OnTimestampValidatedHandler OnTimestampValidated;

public delegate void OnTimestampValidatedHandler(object sender, XAdESSignerTimestampValid

public class XAdESSignerTimestampValidatedEventArgs : EventArgs {
  public int Index { get; }
  public string EntityLabel { get; }
```

```
  public string IssuerRDN { get; }
  public byte[] SerialNumber { get; }
  public byte[] SubjectKeyID { get; }
  public string Time { get; }
  public int ValidationResult { get; }
  public int ChainValidationResult { get; }
  public int ChainValidationDetails { get; }
  public bool Cancel { get; set; }
}
```

Remarks

This event is fired upon the completion of the timestamp validation routine, and reports the respective validation result.

ValidationResult is set to 0 if the validation has been successful, or to a non-zero value in case of a failure.

svtValid	0	The signature is valid
svtUnknown	1	Signature validity is unknown
svtCorrupted	2	The signature is corrupted
svtSignerNotFound	3	Failed to acquire the signing certificate. The signature cannot be validated.
svtFailure	4	General failure
svtReferenceCorrupted	5	Reference corrupted (XML-based signatures only)

TLSCertNeeded Event (XAdESSigner Component)

Fires when a remote TLS party requests a client certificate.

Syntax

C# VB.NET

```
public event OnTLSCertNeededHandler OnTLSCertNeeded;

public delegate void OnTLSCertNeededHandler(object sender, XAdESSignerTLSCertNeededEventA

public class XAdESSignerTLSCertNeededEventArgs : EventArgs {
  public string Host { get; }
  public string CANames { get; }
}
```

Remarks

This event fires to notify the implementation that a remote TLS server has requested a client certificate. The *Host* parameter identifies the host that makes a request, and the *CANames* parameter (optional, according to the TLS spec) advises on the accepted issuing CAs.

Use the TLSClientChain property in response to this event to provide the requested certificate. Please make sure the client certificate includes the associated private key. Note that you may set the certificates before the connection without waiting for this event to fire.

This event is preceded by the TLSHandshake event for the given host and, if the certificate was accepted, succeeded by the TLSEstablished event.

TLSCertValidate Event (XAdESSigner Component)

This event is fired upon receipt of the TLS server's certificate, allowing the user to control its acceptance.

Syntax

C# VB.NET

```
public event OnTLSCertValidateHandler OnTLSCertValidate;

public delegate void OnTLSCertValidateHandler(object sender, XAdESSignerTLSCertValidateEv

public class XAdESSignerTLSCertValidateEventArgs : EventArgs {
  public string ServerHost { get; }
  public string ServerIP { get; }
  public bool Accept { get; set; }
}
```

Remarks

This event is fired during a TLS handshake. Use the TLSServerChain property to access the certificate chain. In general, components may contact a number of TLS endpoints during their work, depending on their configuration.

Accept is assigned in accordance with the outcome of the internal validation check performed by the component, and can be adjusted if needed.

TLSEstablished Event (XAdESSigner Component)

Fires when a TLS handshake with *Host* successfully completes.

Syntax

C# VB.NET

```
public event OnTLSEstablishedHandler OnTLSEstablished;

public delegate void OnTLSEstablishedHandler(object sender, XAdESSignerTLSEstablishedEver

public class XAdESSignerTLSEstablishedEventArgs : EventArgs {
  public string Host { get; }
  public string Version { get; }
  public string Ciphersuite { get; }
  public byte[] ConnectionId { get; }
  public bool Abort { get; set; }
}
```

Remarks

The component uses this event to notify the application about a successful completion of a TLS handshake.

The *Version*, *Ciphersuite*, and *ConnectionId* parameters indicate the security parameters of the new connection. Use the *Abort* parameter if you need to terminate the connection at this stage.

TLSHandshake Event (XAdESSigner Component)

Fires when a new TLS handshake is initiated, before the handshake commences.

Syntax

C# VB.NET

```
public event OnTLSHandshakeHandler OnTLSHandshake;

public delegate void OnTLSHandshakeHandler(object sender, XAdESSignerTLSHandshakeEventArg

public class XAdESSignerTLSHandshakeEventArgs : EventArgs {
  public string Host { get; }
  public bool Abort { get; set; }
}
```

Remarks

The component uses this event to notify the application about the start of a new TLS handshake to *Host*. If the handshake is successful, this event will be followed by the TLSEstablished event. If the server chooses to request a client certificate, the TLSCertNeeded event will also be fired.

TLSShutdown Event (XAdESSigner Component)

Reports the graceful closure of a TLS connection.

Syntax

C# VB.NET

```
public event OnTLSShutdownHandler OnTLSShutdown;

public delegate void OnTLSShutdownHandler(object sender, XAdESSignerTLSShutdownEventArgs

public class XAdESSignerTLSShutdownEventArgs : EventArgs {
  public string Host { get; }
}
```

Remarks

This event notifies the application about the closure of an earlier established TLS connection. Note that only graceful connection closures are reported.

Certificate Type

Encapsulates an individual X.509 certificate.

Remarks

This type keeps and provides access to X.509 certificate details.

The following fields are available:

- Bytes
- CA
- CAKeyID
- CertType
- CRLDistributionPoints
- Curve
- Fingerprint
- FriendlyName
- HashAlgorithm
- Issuer
- IssuerRDN
- KeyAlgorithm
- KeyBits
- KeyFingerprint
- KeyUsage
- KeyValid
- OCSPLocations
- OCSPNoCheck
- Origin
- PolicyIDs
- PrivateKeyBytes
- PrivateKeyExists
- PrivateKeyExtractable
- PublicKeyBytes
- Qualified
- QualifiedStatements
- Qualifiers
- SelfSigned
- SerialNumber

- SigAlgorithm
- Source
- Subject
- SubjectAlternativeName
- SubjectKeyID
- SubjectRDN
- Valid
- ValidFrom
- ValidTo

Fields

Bytes

byte[] (read-only)
Default: ""

Returns the raw certificate data in DER format.

CA

bool
Default: false

Indicates whether the certificate has a CA capability. For the certificate to be considered a CA, it must have its Basic Constraints extension set with the CA indicator enabled.

Set this field when generating a new certificate to have its Basic Constraints extension generated automatically.

CAKeyID

byte[] (read-only)
Default: ""

A unique identifier (fingerprint) of the CA certificate's cryptographic key.

Authority Key Identifier is a certificate extension which allows identification of certificates belonging to the same issuer, but with different public keys. It is a de-facto standard to include this extension in all certificates to facilitate chain building.

This setting cannot be set when generating a certificate as it always derives from another certificate property. CertificateManager generates this setting automatically if enough information is available to it: for self-signed certificates, this value is copied from the SubjectKeyID setting, and for lower-level certificates, from the parent certificate's subject key ID extension.

CertType

CertTypes (read-only)

Default: 0

Returns the type of the entity contained in the Certificate object.

A Certificate object can contain two types of cryptographic objects: a ready-to-use X.509 certificate, or a certificate request ("an unsigned certificate"). Certificate requests can be upgraded to full certificates by signing them with a CA certificate.

Use the CertificateManager component to load or create new certificate and certificate requests objects.

CRLDistributionPoints

string

Default: ""

Contains a list of locations of CRL distribution points used to check this certificate's validity. The list is taken from the respective certificate extension.

Use this field when generating a certificate to provide a list of CRL endpoints that should be made part of the new certificate.

The endpoints are provided as a list of CRLF-separated URLs. Note that this differs from the behaviour used in earlier product versions, where the "|" character was used as the location separator.

Curve

string

Default: ""

Specifies the elliptic curve associated with the certificate's public key. This setting only applies to certificates containing EC keys.

SB_EC_SECP112R1	`SECP112R1`
SB_EC_SECP112R2	`SECP112R2`
SB_EC_SECP128R1	`SECP128R1`
SB_EC_SECP128R2	`SECP128R2`
SB_EC_SECP160K1	`SECP160K1`

SB_EC_SECP160R1	`SECP160R1`
SB_EC_SECP160R2	`SECP160R2`
SB_EC_SECP192K1	`SECP192K1`
SB_EC_SECP192R1	`SECP192R1`
SB_EC_SECP224K1	`SECP224K1`
SB_EC_SECP224R1	`SECP224R1`
SB_EC_SECP256K1	`SECP256K1`
SB_EC_SECP256R1	`SECP256R1`
SB_EC_SECP384R1	`SECP384R1`
SB_EC_SECP521R1	`SECP521R1`
SB_EC_SECT113R1	`SECT113R1`
SB_EC_SECT113R2	`SECT113R2`
SB_EC_SECT131R1	`SECT131R1`
SB_EC_SECT131R2	`SECT131R2`
SB_EC_SECT163K1	`SECT163K1`
SB_EC_SECT163R1	`SECT163R1`
SB_EC_SECT163R2	`SECT163R2`
SB_EC_SECT193R1	`SECT193R1`
SB_EC_SECT193R2	`SECT193R2`
SB_EC_SECT233K1	`SECT233K1`
SB_EC_SECT233R1	`SECT233R1`

SB_EC_SECT239K1	`SECT239K1`
SB_EC_SECT283K1	`SECT283K1`
SB_EC_SECT283R1	`SECT283R1`
SB_EC_SECT409K1	`SECT409K1`
SB_EC_SECT409R1	`SECT409R1`
SB_EC_SECT571K1	`SECT571K1`
SB_EC_SECT571R1	`SECT571R1`
SB_EC_PRIME192V1	`PRIME192V1`
SB_EC_PRIME192V2	`PRIME192V2`
SB_EC_PRIME192V3	`PRIME192V3`
SB_EC_PRIME239V1	`PRIME239V1`
SB_EC_PRIME239V2	`PRIME239V2`
SB_EC_PRIME239V3	`PRIME239V3`
SB_EC_PRIME256V1	`PRIME256V1`
SB_EC_C2PNB163V1	`C2PNB163V1`
SB_EC_C2PNB163V2	`C2PNB163V2`
SB_EC_C2PNB163V3	`C2PNB163V3`
SB_EC_C2PNB176W1	`C2PNB176W1`
SB_EC_C2TNB191V1	`C2TNB191V1`
SB_EC_C2TNB191V2	`C2TNB191V2`
SB_EC_C2TNB191V3	`C2TNB191V3`

SB_EC_C2ONB191V4	`C2ONB191V4`
SB_EC_C2ONB191V5	`C2ONB191V5`
SB_EC_C2PNB208W1	`C2PNB208W1`
SB_EC_C2TNB239V1	`C2TNB239V1`
SB_EC_C2TNB239V2	`C2TNB239V2`
SB_EC_C2TNB239V3	`C2TNB239V3`
SB_EC_C2ONB239V4	`C2ONB239V4`
SB_EC_C2ONB239V5	`C2ONB239V5`
SB_EC_C2PNB272W1	`C2PNB272W1`
SB_EC_C2PNB304W1	`C2PNB304W1`
SB_EC_C2TNB359V1	`C2TNB359V1`
SB_EC_C2PNB368W1	`C2PNB368W1`
SB_EC_C2TNB431R1	`C2TNB431R1`
SB_EC_NISTP192	`NISTP192`
SB_EC_NISTP224	`NISTP224`
SB_EC_NISTP256	`NISTP256`
SB_EC_NISTP384	`NISTP384`
SB_EC_NISTP521	`NISTP521`
SB_EC_NISTB163	`NISTB163`
SB_EC_NISTB233	`NISTB233`
SB_EC_NISTB283	`NISTB283`

SB_EC_NISTB409	`NISTB409`
SB_EC_NISTB571	`NISTB571`
SB_EC_NISTK163	`NISTK163`
SB_EC_NISTK233	`NISTK233`
SB_EC_NISTK283	`NISTK283`
SB_EC_NISTK409	`NISTK409`
SB_EC_NISTK571	`NISTK571`
SB_EC_GOSTCPTEST	`GOSTCPTEST`
SB_EC_GOSTCPA	`GOSTCPA`
SB_EC_GOSTCPB	`GOSTCPB`
SB_EC_GOSTCPC	`GOSTCPC`
SB_EC_GOSTCPXCHA	`GOSTCPXCHA`
SB_EC_GOSTCPXCHB	`GOSTCPXCHB`
SB_EC_BRAINPOOLP160R1	`BRAINPOOLP160R1`
SB_EC_BRAINPOOLP160T1	`BRAINPOOLP160T1`
SB_EC_BRAINPOOLP192R1	`BRAINPOOLP192R1`
SB_EC_BRAINPOOLP192T1	`BRAINPOOLP192T1`
SB_EC_BRAINPOOLP224R1	`BRAINPOOLP224R1`
SB_EC_BRAINPOOLP224T1	`BRAINPOOLP224T1`
SB_EC_BRAINPOOLP256R1	`BRAINPOOLP256R1`
SB_EC_BRAINPOOLP256T1	`BRAINPOOLP256T1`

SB_EC_BRAINPOOLP320R1	`BRAINPOOLP320R1`
SB_EC_BRAINPOOLP320T1	`BRAINPOOLP320T1`
SB_EC_BRAINPOOLP384R1	`BRAINPOOLP384R1`
SB_EC_BRAINPOOLP384T1	`BRAINPOOLP384T1`
SB_EC_BRAINPOOLP512R1	`BRAINPOOLP512R1`
SB_EC_BRAINPOOLP512T1	`BRAINPOOLP512T1`
SB_EC_CURVE25519	`CURVE25519`
SB_EC_CURVE448	`CURVE448`

Fingerprint

string (read-only)
Default: ""

Contains the fingerprint (a hash imprint) of this certificate.

While there is no formal standard defining what a fingerprint is, a SHA1 hash of the certificate's DER-encoded body is typically used.

FriendlyName

string (read-only)
Default: ""

Contains an associated alias (friendly name) of the certificate. The friendly name is not a property of a certificate: it is maintained by the certificate media rather than being included in its DER representation. Windows certificate stores are one example of media that does support friendly names.

HashAlgorithm

string
Default: ""

Provides means to set the hash algorithm to be used in the subsequent operation on the certificate (such as generation or key signing). It is not a property of a certificate; use SigAlgorithm to find out the hash algorithm that is part of the certificate signature.

SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_MD2	`MD2`
SB_HASH_ALGORITHM_MD4	`MD4`
SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_CRC32	`CRC32`
SB_HASH_ALGORITHM_SSL3	`SSL3`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_POLY1305	`POLY1305`
SB_HASH_ALGORITHM_SHA3_224	`SHA3_224`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`
SB_HASH_ALGORITHM_BLAKE2S_128	`BLAKE2S_128`
SB_HASH_ALGORITHM_BLAKE2S_160	`BLAKE2S_160`
SB_HASH_ALGORITHM_BLAKE2S_224	`BLAKE2S_224`

SB_HASH_ALGORITHM_BLAKE2S_256	`BLAKE2S_256`
SB_HASH_ALGORITHM_BLAKE2B_160	`BLAKE2B_160`
SB_HASH_ALGORITHM_BLAKE2B_256	`BLAKE2B_256`
SB_HASH_ALGORITHM_BLAKE2B_384	`BLAKE2B_384`
SB_HASH_ALGORITHM_BLAKE2B_512	`BLAKE2B_512`
SB_HASH_ALGORITHM_SHAKE_128	`SHAKE_128`
SB_HASH_ALGORITHM_SHAKE_256	`SHAKE_256`
SB_HASH_ALGORITHM_SHAKE_128_LEN	`SHAKE_128_LEN`
SB_HASH_ALGORITHM_SHAKE_256_LEN	`SHAKE_256_LEN`

Issuer

string (read-only)
Default: ""

The common name of the certificate issuer (CA), typically a company name. This is part of a larger set of credentials available via IssuerRDN.

IssuerRDN

string
Default: ""

A list of `Property=Value` pairs that uniquely identify the certificate issuer.

Example: */C=US/O=Nationwide CA/CN=Web Certification Authority*

KeyAlgorithm

string
Default: "0"

Specifies the public key algorithm of this certificate.

SB_CERT_ALGORITHM_ID_RSA_ENCRYPTION	`rsaEncryption`
SB_CERT_ALGORITHM_MD2_RSA_ENCRYPTION	`md2withRSAEncryption`
SB_CERT_ALGORITHM_MD5_RSA_ENCRYPTION	`md5withRSAEncryption`
SB_CERT_ALGORITHM_SHA1_RSA_ENCRYPTION	`sha1withRSAEncryption`
SB_CERT_ALGORITHM_ID_DSA	`id-dsa`
SB_CERT_ALGORITHM_ID_DSA_SHA1	`id-dsa-with-sha1`
SB_CERT_ALGORITHM_DH_PUBLIC	`dhpublicnumber`
SB_CERT_ALGORITHM_SHA224_RSA_ENCRYPTION	`sha224WithRSAEncryption`
SB_CERT_ALGORITHM_SHA256_RSA_ENCRYPTION	`sha256WithRSAEncryption`
SB_CERT_ALGORITHM_SHA384_RSA_ENCRYPTION	`sha384WithRSAEncryption`
SB_CERT_ALGORITHM_SHA512_RSA_ENCRYPTION	`sha512WithRSAEncryption`
SB_CERT_ALGORITHM_ID_RSAPSS	`id-RSASSA-PSS`
SB_CERT_ALGORITHM_ID_RSAOAEP	`id-RSAES-OAEP`
SB_CERT_ALGORITHM_RSASIGNATURE_RIPEMD160	`ripemd160withRSA`
SB_CERT_ALGORITHM_ID_ELGAMAL	`elGamal`
SB_CERT_ALGORITHM_SHA1_ECDSA	`ecdsa-with-SHA1`
SB_CERT_ALGORITHM_RECOMMENDED_ECDSA	`ecdsa-recommended`
SB_CERT_ALGORITHM_SHA224_ECDSA	`ecdsa-with-SHA224`
SB_CERT_ALGORITHM_SHA256_ECDSA	`ecdsa-with-SHA256`
SB_CERT_ALGORITHM_SHA384_ECDSA	`ecdsa-with-SHA384`
SB_CERT_ALGORITHM_SHA512_ECDSA	`ecdsa-with-SHA512`

SB_CERT_ALGORITHM_EC	`id-ecPublicKey`
SB_CERT_ALGORITHM_SPECIFIED_ECDSA	`ecdsa-specified`
SB_CERT_ALGORITHM_GOST_R3410_1994	`id-GostR3410-94`
SB_CERT_ALGORITHM_GOST_R3410_2001	`id-GostR3410-2001`
SB_CERT_ALGORITHM_GOST_R3411_WITH_R3410_1994	`id-GostR3411-94-with-GostR3410-94`
SB_CERT_ALGORITHM_GOST_R3411_WITH_R3410_2001	`id-GostR3411-94-with-GostR3410-200`
SB_CERT_ALGORITHM_SHA1_ECDSA_PLAIN	`ecdsa-plain-SHA1`
SB_CERT_ALGORITHM_SHA224_ECDSA_PLAIN	`ecdsa-plain-SHA224`
SB_CERT_ALGORITHM_SHA256_ECDSA_PLAIN	`ecdsa-plain-SHA256`
SB_CERT_ALGORITHM_SHA384_ECDSA_PLAIN	`ecdsa-plain-SHA384`
SB_CERT_ALGORITHM_SHA512_ECDSA_PLAIN	`ecdsa-plain-SHA512`
SB_CERT_ALGORITHM_RIPEMD160_ECDSA_PLAIN	`ecdsa-plain-RIPEMD160`
SB_CERT_ALGORITHM_WHIRLPOOL_RSA_ENCRYPTION	`whirlpoolWithRSAEncryption`
SB_CERT_ALGORITHM_ID_DSA_SHA224	`id-dsa-with-sha224`
SB_CERT_ALGORITHM_ID_DSA_SHA256	`id-dsa-with-sha256`
SB_CERT_ALGORITHM_SHA3_224_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-224`
SB_CERT_ALGORITHM_SHA3_256_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-256`
SB_CERT_ALGORITHM_SHA3_384_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-384`
SB_CERT_ALGORITHM_SHA3_512_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-512`
SB_CERT_ALGORITHM_SHA3_224_ECDSA	`id-ecdsa-with-sha3-224`
SB_CERT_ALGORITHM_SHA3_256_ECDSA	`id-ecdsa-with-sha3-256`

SB_CERT_ALGORITHM_SHA3_384_ECDSA	`id-ecdsa-with-sha3-384`
SB_CERT_ALGORITHM_SHA3_512_ECDSA	`id-ecdsa-with-sha3-512`
SB_CERT_ALGORITHM_SHA3_224_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-224`
SB_CERT_ALGORITHM_SHA3_256_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-256`
SB_CERT_ALGORITHM_SHA3_384_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-384`
SB_CERT_ALGORITHM_SHA3_512_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-512`
SB_CERT_ALGORITHM_ID_DSA_SHA3_224	`id-dsa-with-sha3-224`
SB_CERT_ALGORITHM_ID_DSA_SHA3_256	`id-dsa-with-sha3-256`
SB_CERT_ALGORITHM_BLAKE2S_128_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s1`
SB_CERT_ALGORITHM_BLAKE2S_160_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s1`
SB_CERT_ALGORITHM_BLAKE2S_224_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s2`
SB_CERT_ALGORITHM_BLAKE2S_256_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s2`
SB_CERT_ALGORITHM_BLAKE2B_160_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b1`
SB_CERT_ALGORITHM_BLAKE2B_256_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b2`
SB_CERT_ALGORITHM_BLAKE2B_384_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b3`
SB_CERT_ALGORITHM_BLAKE2B_512_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b5`
SB_CERT_ALGORITHM_BLAKE2S_128_ECDSA	`id-ecdsa-with-blake2s128`
SB_CERT_ALGORITHM_BLAKE2S_160_ECDSA	`id-ecdsa-with-blake2s160`
SB_CERT_ALGORITHM_BLAKE2S_224_ECDSA	`id-ecdsa-with-blake2s224`
SB_CERT_ALGORITHM_BLAKE2S_256_ECDSA	`id-ecdsa-with-blake2s256`
SB_CERT_ALGORITHM_BLAKE2B_160_ECDSA	`id-ecdsa-with-blake2b160`

SB_CERT_ALGORITHM_BLAKE2B_256_ECDSA	`id-ecdsa-with-blake2b256`
SB_CERT_ALGORITHM_BLAKE2B_384_ECDSA	`id-ecdsa-with-blake2b384`
SB_CERT_ALGORITHM_BLAKE2B_512_ECDSA	`id-ecdsa-with-blake2b512`
SB_CERT_ALGORITHM_BLAKE2S_128_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s128`
SB_CERT_ALGORITHM_BLAKE2S_160_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s160`
SB_CERT_ALGORITHM_BLAKE2S_224_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s224`
SB_CERT_ALGORITHM_BLAKE2S_256_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s256`
SB_CERT_ALGORITHM_BLAKE2B_160_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b160`
SB_CERT_ALGORITHM_BLAKE2B_256_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b256`
SB_CERT_ALGORITHM_BLAKE2B_384_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b384`
SB_CERT_ALGORITHM_BLAKE2B_512_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b512`
SB_CERT_ALGORITHM_ID_DSA_BLAKE2S_224	`id-dsa-with-blake2s224`
SB_CERT_ALGORITHM_ID_DSA_BLAKE2S_256	`id-dsa-with-blake2s256`
SB_CERT_ALGORITHM_EDDSA_ED25519	`id-Ed25519`
SB_CERT_ALGORITHM_EDDSA_ED448	`id-Ed448`
SB_CERT_ALGORITHM_EDDSA_ED25519_PH	`id-Ed25519ph`
SB_CERT_ALGORITHM_EDDSA_ED448_PH	`id-Ed448ph`
SB_CERT_ALGORITHM_EDDSA	`id-EdDSA`
SB_CERT_ALGORITHM_EDDSA_SIGNATURE	`id-EdDSA-sig`
SB_CERT_ALGORITHM_MLDSA_44	`id-ml-dsa-44`
SB_CERT_ALGORITHM_MLDSA_65	`id-ml-dsa-65`

SB_CERT_ALGORITHM_MLDSA_87	`id-ml-dsa-87`
SB_CERT_ALGORITHM_HASH_MLDSA_44_SHA512	`id-hash-ml-dsa-44-with-sha512`
SB_CERT_ALGORITHM_HASH_MLDSA_65_SHA512	`id-hash-ml-dsa-65-with-sha512`
SB_CERT_ALGORITHM_HASH_MLDSA_87_SHA512	`id-hash-ml-dsa-87-with-sha512`

Use the KeyBits, Curve, and PublicKeyBytes fields to get more details about the key the certificate contains.

KeyBits

int (read-only)
Default: 0

Returns the length of the public key in bits.

This value indicates the length of the principal cryptographic parameter of the key, such as the length of the RSA modulus or ECDSA field. The key data returned by the PublicKeyBytes or PrivateKeyBytes field would typically contain auxiliary values, and therefore be longer.

KeyFingerprint

string (read-only)
Default: ""

Returns a SHA1 fingerprint of the public key contained in the certificate.

Note that the key fingerprint is different from the certificate fingerprint accessible via the Fingerprint field. The key fingeprint uniquely identifies the public key, and so can be the same for multiple certificates containing the same key.

KeyUsage

int
Default: 0

Indicates the purposes of the key contained in the certificate, in the form of an OR'ed flag set.

This value is a bit mask of the following values:

ckuUnknown	`0x00000`	Unknown key usage

ckuDigitalSignature	`0x00001`	Digital signature
ckuNonRepudiation	`0x00002`	Non-repudiation
ckuKeyEncipherment	`0x00004`	Key encipherment
ckuDataEncipherment	`0x00008`	Data encipherment
ckuKeyAgreement	`0x00010`	Key agreement
ckuKeyCertSign	`0x00020`	Certificate signing
ckuCRLSign	`0x00040`	Revocation signing
ckuEncipherOnly	`0x00080`	Encipher only
ckuDecipherOnly	`0x00100`	Decipher only
ckuServerAuthentication	`0x00200`	Server authentication
ckuClientAuthentication	`0x00400`	Client authentication
ckuCodeSigning	`0x00800`	Code signing
ckuEmailProtection	`0x01000`	Email protection
ckuTimeStamping	`0x02000`	Timestamping
ckuOCSPSigning	`0x04000`	OCSP signing
ckuSmartCardLogon	`0x08000`	Smartcard logon
ckuKeyPurposeClientAuth	`0x10000`	Kerberos - client authentication
ckuKeyPurposeKDC	`0x20000`	Kerberos - KDC

Set this field before generating the certificate to propagate the key usage flags to the new certificate.

KeyValid

bool (read-only)

Default: false

Returns `True` if the certificate's key is cryptographically valid, and `False` otherwise.

OCSPLocations

string

Default: ""

Locations of OCSP services that can be used to check this certificate's validity in real time, as recorded by the CA.

Set this field before calling the certificate manager's Generate method to propagate it to the new certificate.

The OCSP locations are provided as a list of CRLF-separated URLs. Note that this differs from the behaviour used in earlier product versions, where the "|" character was used as the location separator.

OCSPNoCheck

bool

Default: false

Accessor to the value of the certificate's ocsp-no-check extension.

Origin

int (read-only)

Default: 0

Returns the location that the certificate was taken or loaded from.

PolicyIDs

string

Default: ""

Contains identifiers (OIDs) of the applicable certificate policies.

The Certificate Policies extension identifies a sequence of policies under which the certificate has been issued, and which regulate its usage.

Set this field when generating a certificate to propagate the policies information to the new certificate.

The policies are provided as a list of CRLF-separated entries. Note that this differs from the behaviour used in earlier product versions, where the "|" character was used as the policy element separator.

PrivateKeyBytes

byte[] (read-only)
Default: ""

Returns the certificate's private key in DER-encoded format. It is normal for this field to be empty if the private key is non-exportable, which, for example, is typical for certificates originating from hardware security devices.

PrivateKeyExists

bool (read-only)
Default: false

Indicates whether the certificate has a usable private key associated with it. If it is set to `True`, the certificate can be used for private key operations, such as signing or decryption.

This field is independent from PrivateKeyBytes, and can be set to `True` even if the former is empty. This would imply that the private key is non-exportable, but still can be used for cryptographic operations.

PrivateKeyExtractable

bool (read-only)
Default: false

Indicates whether the private key is extractable (exportable).

PublicKeyBytes

byte[] (read-only)
Default: ""

Contains the certificate's public key in DER format.

This typically would contain an ASN.1-encoded public key value. The exact format depends on the type of the public key contained in the certificate.

Qualified

bool (read-only)

Default: false

Indicates whether the certificate is qualified.

This property is set to `True` if the certificate is confirmed by a Trusted List to be qualified.

QualifiedStatements

QualifiedStatementsTypes

Default: 0

Returns a simplified qualified status of the certificate.

Qualifiers

string (read-only)

Default: ""

A list of qualifiers.

Contains a comma-separated list of qualifier aliases for the certificate, for example *QCP-n-qscd,QCWithSSCD*.

SelfSigned

bool (read-only)

Default: false

Indicates whether the certificate is self-signed (root) or signed by an external CA.

SerialNumber

byte[]

Default: ""

Returns the certificate's serial number.

The serial number is a binary string that uniquely identifies a certificate among others issued by the same CA. According to the X.509 standard, the (issuer, serial number) pair should be globally unique to facilitate chain building.

SigAlgorithm

string (read-only)

Default: ""

Indicates the algorithm that was used by the CA to sign this certificate.

A signature algorithm typically combines hash and public key algorithms together, such as `sha256WithRSAEncryption` or `ecdsa-with-SHA256`.

Source

PKISources (read-only)
Default: 0

Returns the source (location or disposition) of a cryptographic primitive entity, such as a certificate, CRL, or OCSP response.

Subject

string (read-only)
Default: ""

The common name of the certificate holder, typically an individual's name, a URL, an e-mail address, or a company name. This is part of a larger set of credentials available via SubjectRDN.

SubjectAlternativeName

string
Default: ""

Returns or sets the value of the Subject Alternative Name extension of the certificate.

Subject alternative names are used to provide additional names that are impractical to store in the main SubjectRDN field. For example, it is often used to store all the domain names that a TLS certificate is authorized to protect.

The alternative names are provided as a list of CRLF-separated entries. Note that this differs from the behaviour used in earlier product versions, where the "|" character was used as the element separator.

SubjectKeyID

byte[]
Default: ""

Contains a unique identifier of the certificate's cryptographic key.

Subject Key Identifier is a certificate extension which allows a specific public key to be associated with a certificate holder. Typically, subject key identifiers of CA certificates are recorded as respective CA key identifiers in the subordinate certificates that they issue, which facilitates chain building.

The SubjectKeyID and CAKeyID fields of self-signed certificates typically contain identical values, as in that specific case, the issuer and the subject are the same entity.

SubjectRDN

string
Default: ""

A list of `Property=Value` pairs that uniquely identify the certificate holder (subject).

Depending on the purpose of the certificate and the policies of the CA that issued it, the values included in the subject record may differ drastically and contain business or personal names, web URLs, email addresses, and other data.

Example: */C=US/O=Oranges and Apples, Inc./OU=Accounts Receivable/1.2.3.4.5=Value with unknown OID/CN=Margaret Watkins*.

Valid

bool (read-only)
Default: false

Indicates whether or not the signature over the certificate or the request is valid and matches the public key contained in the CA certificate/request.

ValidFrom

string
Default: ""

The time point at which the certificate becomes valid, in UTC.

ValidTo

string
Default: ""

The time point at which the certificate expires, in UTC.

Constructors

C# VB.NET

```
public Certificate(byte[] bytes, int startIndex, int count, string password);
```

Loads the X.509 certificate from a memory buffer. *Bytes* is a buffer containing the raw certificate data. *StartIndex* and *Count* specify the starting position and number of bytes to be read from the buffer, respectively. *Password* is a password encrypting the certificate.

C# VB.NET

```
public Certificate(byte[] certBytes, int certStartIndex, int certCount, byte[] keyBytes,
```

Loads the X.509 certificate from a memory buffer.

CertBytes is a buffer containing the raw certificate data. *CertStartIndex* and *CertCount* specify the starting position and number of bytes to be read from the buffer, respectively.

KeyBytes is a buffer containing the private key data. *KeyStartIndex* and *KeyCount* specify the starting position and number of bytes to be read from the buffer, respectively.

Password is a password encrypting the certificate.

C# VB.NET

```
public Certificate(byte[] bytes, int startIndex, int count);
```

Loads the X.509 certificate from a memory buffer. *Bytes* is a buffer containing the raw certificate data. *StartIndex* and *Count* specify the starting position and number of bytes to be read from the buffer, respectively.

C# VB.NET

```
public Certificate(string path, string password);
```

Loads the X.509 certificate from a file. *Path* specifies the full path to the file containing the certificate data. *Password* is a password encrypting the certificate.

C# VB.NET

```
public Certificate(string certPath, string keyPath, string password);
```

Loads the X.509 certificate from a file. *CertPath* specifies the full path to the file containing the certificate data. *KeyPath* specifies the full path to the file containing the private key. *Password* is a password encrypting the certificate.

C# VB.NET

```
public Certificate(string path);
```

Loads the X.509 certificate from a file. *Path* specifies the full path to the file containing the certificate data.

C# VB.NET

```
public Certificate(System.IO.Stream stream);
```

Loads the X.509 certificate from a stream. *Stream* is a stream containing the certificate data.

C# VB.NET

```
public Certificate(System.IO.Stream stream, string password);
```

Loads the X.509 certificate from a stream. *Stream* is a stream containing the certificate data. *Password* is a password encrypting the certificate.

C# VB.NET

```
public Certificate(System.IO.Stream certStream, System.IO.Stream keyStream, string passwo
```

Loads the X.509 certificate from a stream. *CertStream* is a stream containing the certificate data. *KeyStream* is a stream containing the private key. *Password* is a password encrypting the certificate.

C# VB.NET

```
public Certificate();
```

Creates a new object with default field values.

CRL Type

Represents a Certificate Revocation List.

Remarks

CRLs store information about revoked certificates, i.e., certificates that have been identified as invalid by their issuing certificate authority (CA) for any number of reasons.

Each CRL object lists certificates from a single CA and identifies them by their serial numbers. A CA may or may not publish a CRL, may publish several CRLs, or may publish the same CRL in multiple locations.

Unlike OCSP responses, CRLs only list certificates that have been revoked. They do not list certificates that are still valid.

The following fields are available:

- Bytes
- CAKeyID
- EntryCount
- Issuer
- IssuerRDN
- Location
- NextUpdate
- SigAlgorithm
- Source
- TBS
- ThisUpdate

Fields

Bytes

byte[] (read-only)
Default: ""

Returns the raw CRL data in DER format.

CAKeyID

byte[]
Default: ""

A unique identifier (fingerprint) of the CA certificate's private key, if present in the CRL.

EntryCount

int (read-only)
Default: 0

Returns the number of certificate status entries in the CRL.

Issuer

string (read-only)
Default: ""

The common name of the CRL issuer (CA), typically a company name.

IssuerRDN
string (read-only)
Default: ""

A collection of information, in the form of [OID, Value] pairs, uniquely identifying the CRL issuer.

Location
string (read-only)
Default: ""

The URL that the CRL was downloaded from.

NextUpdate
string
Default: ""

The planned time and date of the next version of this CRL to be published.

SigAlgorithm
string
Default: "0"

The public key algorithm that was used by the CA to sign this CRL.

Source
PKISources (read-only)
Default: 0

Returns the source (location or disposition) of a cryptographic primitive entity, such as a certificate, CRL, or OCSP response.

TBS
byte[] (read-only)
Default: ""

The to-be-signed part of the CRL (the CRL without the signature part).

ThisUpdate
string
Default: ""

The date and time at which this version of the CRL was published.

Constructors

C# VB.NET

```
public CRL(byte[] bytes, int startIndex, int count);
```

Creates a CRL object from a memory buffer. *Bytes* is a buffer containing raw (DER) CRL data, *StartIndex* and *Count* specify the starting position and the length of the CRL data in the buffer, respectively.

C# VB.NET

```
public CRL(string location);
```

Creates a CRL object by downloading it from a remote location.

C# VB.NET

```
public CRL(System.IO.Stream stream);
```

Creates a CRL object from data contained in a stream.

C# VB.NET

```
public CRL();
```

Creates an empty CRL object.

ExternalCrypto Type

Specifies the parameters of external cryptographic calls.

Remarks

External cryptocalls are used in a Distributed Cryptography (DC) subsystem, which allows the delegation of security operations to the remote agent. For instance, it can be used to compute the signature value on the server, while retaining the client's private key locally.

The following fields are available:

- AsyncDocumentID
- CustomParams

- Data
- ExternalHashCalculation
- HashAlgorithm
- KeyID
- KeySecret
- Method
- Mode
- PublicKeyAlgorithm

Fields

AsyncDocumentID

string

Default: ""

Specifies an optional document ID for SignAsyncBegin() and SignAsyncEnd() calls.

Use this property when working with multi-signature DCAuth requests and responses to uniquely identify documents signed within a larger batch. On the completion stage, this value helps the signing component identify the correct signature in the returned batch of responses.

If using batched requests, make sure to set this property to the same value on both the pre-signing (SignAsyncBegin) and completion (SignAsyncEnd) stages.

CustomParams

string

Default: ""

Custom parameters to be passed to the signing service (uninterpreted).

Data

string

Default: ""

Additional data to be included in the async state and mirrored back by the requestor.

ExternalHashCalculation

bool

Default: false

Specifies whether the message hash is to be calculated at the external endpoint. Please note that this mode is not supported by the DCAuth component.

If set to true, the component will pass a few kilobytes of to-be-signed data from the document to the OnExternalSign event. This only applies when SignExternal() is called.

HashAlgorithm

string

Default: "sha256"

Specifies the request's signature hash algorithm.

SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_MD2	`MD2`
SB_HASH_ALGORITHM_MD4	`MD4`
SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_CRC32	`CRC32`
SB_HASH_ALGORITHM_SSL3	`SSL3`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_POLY1305	`POLY1305`
SB_HASH_ALGORITHM_SHA3_224	`SHA3_224`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`

SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`
SB_HASH_ALGORITHM_BLAKE2S_128	`BLAKE2S_128`
SB_HASH_ALGORITHM_BLAKE2S_160	`BLAKE2S_160`
SB_HASH_ALGORITHM_BLAKE2S_224	`BLAKE2S_224`
SB_HASH_ALGORITHM_BLAKE2S_256	`BLAKE2S_256`
SB_HASH_ALGORITHM_BLAKE2B_160	`BLAKE2B_160`
SB_HASH_ALGORITHM_BLAKE2B_256	`BLAKE2B_256`
SB_HASH_ALGORITHM_BLAKE2B_384	`BLAKE2B_384`
SB_HASH_ALGORITHM_BLAKE2B_512	`BLAKE2B_512`
SB_HASH_ALGORITHM_SHAKE_128	`SHAKE_128`
SB_HASH_ALGORITHM_SHAKE_256	`SHAKE_256`
SB_HASH_ALGORITHM_SHAKE_128_LEN	`SHAKE_128_LEN`
SB_HASH_ALGORITHM_SHAKE_256_LEN	`SHAKE_256_LEN`

KeyID

string

Default: ""

The ID of the pre-shared key used for DC request authentication.

Asynchronous DCAuth-driven communication requires that parties authenticate each other with a secret pre-shared cryptographic key. This provides an extra protection layer for the protocol and diminishes the risk of the private key becoming abused by foreign parties. Use this property to provide the pre-shared key identifier, and use KeySecret to pass the key itself.

The same KeyID/KeySecret pair should be used on the DCAuth side for the signing requests to be accepted.

Note: The KeyID/KeySecret scheme is very similar to the AuthKey scheme used in various Cloud service providers to authenticate users.

Example:

Copy

```
signer.ExternalCrypto.KeyID = "MainSigningKey";
signer.ExternalCrypto.KeySecret = "abcdef0123456789";
```

KeySecret

string
Default: ""

The pre-shared key used for DC request authentication. This key must be set and match the key used by the DCAuth counterpart for the scheme to work.

Read more about configuring authentication in the KeyID topic.

Method

AsyncSignMethods
Default: 0

Specifies the asynchronous signing method. This is typically defined by the DC server capabilities and setup.

Available options:

asmdPKCS1	`0`
asmdPKCS7	`1`

Mode

ExternalCryptoModes
Default: 0

Specifies the external cryptography mode.

Available options:

ecmDefault	The default value (`0`)
ecmDisabled	Do not use DC or external signing (`1`)
ecmGeneric	Generic external signing with the OnExternalSign event (`2`)
ecmDCAuth	DCAuth signing (`3`)
ecmDCAuthJSON	DCAuth signing in JSON format (`4`)

PublicKeyAlgorithm

string

Default: ""

Provide the public key algorithm here if the certificate is not available on the pre-signing stage.

SB_CERT_ALGORITHM_ID_RSA_ENCRYPTION	`rsaEncryption`
SB_CERT_ALGORITHM_MD2_RSA_ENCRYPTION	`md2withRSAEncryption`
SB_CERT_ALGORITHM_MD5_RSA_ENCRYPTION	`md5withRSAEncryption`
SB_CERT_ALGORITHM_SHA1_RSA_ENCRYPTION	`sha1withRSAEncryption`
SB_CERT_ALGORITHM_ID_DSA	`id-dsa`
SB_CERT_ALGORITHM_ID_DSA_SHA1	`id-dsa-with-sha1`
SB_CERT_ALGORITHM_DH_PUBLIC	`dhpublicnumber`
SB_CERT_ALGORITHM_SHA224_RSA_ENCRYPTION	`sha224WithRSAEncryption`
SB_CERT_ALGORITHM_SHA256_RSA_ENCRYPTION	`sha256WithRSAEncryption`
SB_CERT_ALGORITHM_SHA384_RSA_ENCRYPTION	`sha384WithRSAEncryption`
SB_CERT_ALGORITHM_SHA512_RSA_ENCRYPTION	`sha512WithRSAEncryption`
SB_CERT_ALGORITHM_ID_RSAPSS	`id-RSASSA-PSS`

SB_CERT_ALGORITHM_ID_RSAOAEP	`id-RSAES-OAEP`
SB_CERT_ALGORITHM_RSASIGNATURE_RIPEMD160	`ripemd160withRSA`
SB_CERT_ALGORITHM_ID_ELGAMAL	`elGamal`
SB_CERT_ALGORITHM_SHA1_ECDSA	`ecdsa-with-SHA1`
SB_CERT_ALGORITHM_RECOMMENDED_ECDSA	`ecdsa-recommended`
SB_CERT_ALGORITHM_SHA224_ECDSA	`ecdsa-with-SHA224`
SB_CERT_ALGORITHM_SHA256_ECDSA	`ecdsa-with-SHA256`
SB_CERT_ALGORITHM_SHA384_ECDSA	`ecdsa-with-SHA384`
SB_CERT_ALGORITHM_SHA512_ECDSA	`ecdsa-with-SHA512`
SB_CERT_ALGORITHM_EC	`id-ecPublicKey`
SB_CERT_ALGORITHM_SPECIFIED_ECDSA	`ecdsa-specified`
SB_CERT_ALGORITHM_GOST_R3410_1994	`id-GostR3410-94`
SB_CERT_ALGORITHM_GOST_R3410_2001	`id-GostR3410-2001`
SB_CERT_ALGORITHM_GOST_R3411_WITH_R3410_1994	`id-GostR3411-94-with-GostR3410-94`
SB_CERT_ALGORITHM_GOST_R3411_WITH_R3410_2001	`id-GostR3411-94-with-GostR3410-200`
SB_CERT_ALGORITHM_SHA1_ECDSA_PLAIN	`ecdsa-plain-SHA1`
SB_CERT_ALGORITHM_SHA224_ECDSA_PLAIN	`ecdsa-plain-SHA224`
SB_CERT_ALGORITHM_SHA256_ECDSA_PLAIN	`ecdsa-plain-SHA256`
SB_CERT_ALGORITHM_SHA384_ECDSA_PLAIN	`ecdsa-plain-SHA384`
SB_CERT_ALGORITHM_SHA512_ECDSA_PLAIN	`ecdsa-plain-SHA512`
SB_CERT_ALGORITHM_RIPEMD160_ECDSA_PLAIN	`ecdsa-plain-RIPEMD160`

SB_CERT_ALGORITHM_WHIRLPOOL_RSA_ENCRYPTION	`whirlpoolWithRSAEncryption`
SB_CERT_ALGORITHM_ID_DSA_SHA224	`id-dsa-with-sha224`
SB_CERT_ALGORITHM_ID_DSA_SHA256	`id-dsa-with-sha256`
SB_CERT_ALGORITHM_SHA3_224_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-224`
SB_CERT_ALGORITHM_SHA3_256_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-256`
SB_CERT_ALGORITHM_SHA3_384_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-384`
SB_CERT_ALGORITHM_SHA3_512_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-sha3-512`
SB_CERT_ALGORITHM_SHA3_224_ECDSA	`id-ecdsa-with-sha3-224`
SB_CERT_ALGORITHM_SHA3_256_ECDSA	`id-ecdsa-with-sha3-256`
SB_CERT_ALGORITHM_SHA3_384_ECDSA	`id-ecdsa-with-sha3-384`
SB_CERT_ALGORITHM_SHA3_512_ECDSA	`id-ecdsa-with-sha3-512`
SB_CERT_ALGORITHM_SHA3_224_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-224`
SB_CERT_ALGORITHM_SHA3_256_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-256`
SB_CERT_ALGORITHM_SHA3_384_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-384`
SB_CERT_ALGORITHM_SHA3_512_ECDSA_PLAIN	`id-ecdsa-plain-with-sha3-512`
SB_CERT_ALGORITHM_ID_DSA_SHA3_224	`id-dsa-with-sha3-224`
SB_CERT_ALGORITHM_ID_DSA_SHA3_256	`id-dsa-with-sha3-256`
SB_CERT_ALGORITHM_BLAKE2S_128_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s1`
SB_CERT_ALGORITHM_BLAKE2S_160_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s1`
SB_CERT_ALGORITHM_BLAKE2S_224_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s2`
SB_CERT_ALGORITHM_BLAKE2S_256_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2s2`

SB_CERT_ALGORITHM_BLAKE2B_160_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b16`
SB_CERT_ALGORITHM_BLAKE2B_256_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b25`
SB_CERT_ALGORITHM_BLAKE2B_384_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b38`
SB_CERT_ALGORITHM_BLAKE2B_512_RSA_ENCRYPTION	`id-rsassa-pkcs1-v1_5-with-blake2b51`
SB_CERT_ALGORITHM_BLAKE2S_128_ECDSA	`id-ecdsa-with-blake2s128`
SB_CERT_ALGORITHM_BLAKE2S_160_ECDSA	`id-ecdsa-with-blake2s160`
SB_CERT_ALGORITHM_BLAKE2S_224_ECDSA	`id-ecdsa-with-blake2s224`
SB_CERT_ALGORITHM_BLAKE2S_256_ECDSA	`id-ecdsa-with-blake2s256`
SB_CERT_ALGORITHM_BLAKE2B_160_ECDSA	`id-ecdsa-with-blake2b160`
SB_CERT_ALGORITHM_BLAKE2B_256_ECDSA	`id-ecdsa-with-blake2b256`
SB_CERT_ALGORITHM_BLAKE2B_384_ECDSA	`id-ecdsa-with-blake2b384`
SB_CERT_ALGORITHM_BLAKE2B_512_ECDSA	`id-ecdsa-with-blake2b512`
SB_CERT_ALGORITHM_BLAKE2S_128_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s128`
SB_CERT_ALGORITHM_BLAKE2S_160_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s160`
SB_CERT_ALGORITHM_BLAKE2S_224_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s224`
SB_CERT_ALGORITHM_BLAKE2S_256_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2s256`
SB_CERT_ALGORITHM_BLAKE2B_160_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b160`
SB_CERT_ALGORITHM_BLAKE2B_256_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b256`
SB_CERT_ALGORITHM_BLAKE2B_384_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b384`
SB_CERT_ALGORITHM_BLAKE2B_512_ECDSA_PLAIN	`id-ecdsa-plain-with-blake2b512`
SB_CERT_ALGORITHM_ID_DSA_BLAKE2S_224	`id-dsa-with-blake2s224`

SB_CERT_ALGORITHM_ID_DSA_BLAKE2S_256	`id-dsa-with-blake2s256`
SB_CERT_ALGORITHM_EDDSA_ED25519	`id-Ed25519`
SB_CERT_ALGORITHM_EDDSA_ED448	`id-Ed448`
SB_CERT_ALGORITHM_EDDSA_ED25519_PH	`id-Ed25519ph`
SB_CERT_ALGORITHM_EDDSA_ED448_PH	`id-Ed448ph`
SB_CERT_ALGORITHM_EDDSA	`id-EdDSA`
SB_CERT_ALGORITHM_EDDSA_SIGNATURE	`id-EdDSA-sig`
SB_CERT_ALGORITHM_MLDSA_44	`id-ml-dsa-44`
SB_CERT_ALGORITHM_MLDSA_65	`id-ml-dsa-65`
SB_CERT_ALGORITHM_MLDSA_87	`id-ml-dsa-87`
SB_CERT_ALGORITHM_HASH_MLDSA_44_SHA512	`id-hash-ml-dsa-44-with-sha512`
SB_CERT_ALGORITHM_HASH_MLDSA_65_SHA512	`id-hash-ml-dsa-65-with-sha512`
SB_CERT_ALGORITHM_HASH_MLDSA_87_SHA512	`id-hash-ml-dsa-87-with-sha512`

Constructors

C# VB.NET

```
public ExternalCrypto();
```

Creates a new ExternalCrypto object with default field values.

OCSPResponse Type

Represents a single OCSP response originating from an OCSP responder.

Remarks

OCSP is a protocol that allows verification of certificate status in real-time, and is an alternative to Certificate Revocation Lists (CRLs).

An OCSP response is a snapshot of the certificate status at a given time.

The following fields are available:

- Bytes
- EntryCount
- Issuer
- IssuerRDN
- Location
- ProducedAt
- SigAlgorithm
- Source

Fields

Bytes

byte[] (read-only)
Default: ""

A buffer containing the raw OCSP response data.

EntryCount

int (read-only)
Default: 0

The number of SingleResponse elements contained in this OCSP response. Each SingleResponse element corresponds to a certificate status.

Issuer

string (read-only)
Default: ""

Indicates the issuer of this response (a CA or its authorized representative).

IssuerRDN

string (read-only)
Default: ""

Indicates the RDN of the issuer of this response (a CA or its authorized representative).

Location

string (read-only)
Default: ""

The location of the OCSP responder.

ProducedAt

string
Default: ""

Specifies the time when the response was produced, in UTC.

SigAlgorithm

string
Default: "0"

The public key algorithm that was used by the CA to sign this OCSP response.

Source

PKISources (read-only)
Default: 0

Returns the source (location or disposition) of a cryptographic primitive entity, such as a certificate, CRL, or OCSP response.

Constructors

C# VB.NET

```
public OCSPResponse(byte[] bytes, int startIndex, int count);
```

Initializes the response from a memory buffer. *Bytes* is a buffer containing raw OCSP response data, *StartIndex* and *Count* specify the starting position and the number of bytes to be read from this buffer.

C# VB.NET

```
public OCSPResponse(string location);
```

Downloads an OCSP response from a remote location.

C# VB.NET

```
public OCSPResponse(System.IO.Stream stream);
```

Initializes the response with the data from a stream.

C# VB.NET

```
public OCSPResponse();
```

Creates an empty OCSP response object.

ProxySettings Type

A container for proxy server settings.

Remarks

This type exposes a collection of properties for tuning up the proxy server configuration.

The following fields are available:

- Address
- Authentication
- Password
- Port
- ProxyType
- RequestHeaders
- ResponseBody
- ResponseHeaders
- UseIPv6
- Username

Fields

Address
string
Default: ""

The IP address of the proxy server.

Authentication

ProxyAuthTypes

Default: 0

The authentication type used by the proxy server.

patNoAuthentication	`0`
patBasic	`1`
patDigest	`2`
patNTLM	`3`

Password

string

Default: ""

The password to authenticate to the proxy server.

Port

int

Default: 0

The port on the proxy server to connect to.

ProxyType

ProxyTypes

Default: 0

The type of the proxy server.

cptNone	`0`
cptSocks4	`1`
cptSocks5	`2`
cptWebTunnel	`3`

cptHTTP	4

RequestHeaders
string
Default: ""

Contains HTTP request headers for WebTunnel and HTTP proxy.

ResponseBody
string
Default: ""

Contains the HTTP or HTTPS (WebTunnel) proxy response body.

ResponseHeaders
string
Default: ""

Contains response headers received from an HTTP or HTTPS (WebTunnel) proxy server.

UseIPv6
bool
Default: false

Specifies whether IPv6 should be used when connecting through the proxy.

Username
string
Default: ""

Specifies the username credential for proxy authentication.

Constructors

C# VB.NET

```
public ProxySettings();
```

Creates a new ProxySettings object.

SocketSettings Type

A container for the socket settings.

Remarks

This type is a container for socket-layer parameters.

The following fields are available:

- DNSMode
- DNSPort
- DNSQueryTimeout
- DNSServers
- DNSTotalTimeout
- IncomingSpeedLimit
- LocalAddress
- LocalPort
- OutgoingSpeedLimit
- Timeout
- UseIPv6

Fields

DNSMode
DNSResolveModes
Default: 0

Selects the DNS resolver to use: the component's (secure) built-in one, or the one provided by the system.

dmAuto	`0`
dmPlatform	`1`
dmOwn	`2`

dmOwnSecure	3

DNSPort
int
Default: 0

Specifies the port number to be used for sending queries to the DNS server.

DNSQueryTimeout
int
Default: 0

The timeout (in milliseconds) for each DNS query. The value of 0 indicates an infinite timeout.

DNSServers
string
Default: ""

The addresses of DNS servers to use for address resolution, separated by commas or semicolons.

DNSTotalTimeout
int
Default: 0

The timeout (in milliseconds) for the whole resolution process. The value of 0 indicates an infinite timeout.

IncomingSpeedLimit
int
Default: 0

The maximum number of bytes to read from the socket, per second.

LocalAddress
string
Default: ""

The local network interface to bind the socket to.

LocalPort

int

Default: 0

The local port number to bind the socket to.

OutgoingSpeedLimit

int

Default: 0

The maximum number of bytes to write to the socket, per second.

Timeout

int

Default: 60000

The maximum period of waiting, in milliseconds, after which the socket operation is considered unsuccessful.

If *Timeout* is set to 0, a socket operation will expire after the system-default timeout (2 hrs 8 min for TCP stack).

UseIPv6

bool

Default: false

Enables or disables IP protocol version 6.

Constructors

C# VB.NET

```
public SocketSettings();
```

Creates a new SocketSettings object.

TimestampInfo Type

A container for timestamp information.

Remarks

The TimestampInfo object contains details of a third-party timestamp and the outcome of its validation.

The following fields are available:

- Accuracy
- Bytes
- CertificateIndex
- ChainValidationDetails
- ChainValidationResult
- ContainsLongTermInfo
- EntityLabel
- HashAlgorithm
- ParentEntity
- SerialNumber
- Time
- TimestampType
- TSAName
- ValidationLog
- ValidationResult

Fields

Accuracy

long (read-only)

Default: 0

This field indicates the accuracy of the included time mark, in microseconds.

Bytes

byte[] (read-only)

Default: ""

Returns the raw timestamp data in DER format.

CertificateIndex

int (read-only)

Default: -1

Returns the index of the TSA certificate in the Certificates collection.

Use this property to look up the TSA certificate in the Certificates collection.

ChainValidationDetails

int (read-only)
Default: 0

The details of a certificate chain validation outcome. They may often suggest the reasons that contributed to the overall validation result.

Returns a bit mask of the following options:

cvrBadData	`0x0001`	One or more certificates in the validation path are malformed
cvrRevoked	`0x0002`	One or more certificates are revoked
cvrNotYetValid	`0x0004`	One or more certificates are not yet valid
cvrExpired	`0x0008`	One or more certificates are expired
cvrInvalidSignature	`0x0010`	A certificate contains a non-valid digital signature
cvrUnknownCA	`0x0020`	A CA certificate for one or more certificates has not been found (chain incomplete)
cvrCAUnauthorized	`0x0040`	One of the CA certificates are not authorized to act as CA
cvrCRLNotVerified	`0x0080`	One or more CRLs could not be verified
cvrOCSPNotVerified	`0x0100`	One or more OCSP responses could not be verified
cvrIdentityMismatch	`0x0200`	The identity protected by the certificate (a TLS endpoint or an e-mail addressee) does not match what is recorded in the certificate
cvrNoKeyUsage	`0x0400`	A mandatory key usage is not enabled in one of the chain certificates

cvrBlocked	`0x0800`	One or more certificates are blocked
cvrFailure	`0x1000`	General validation failure
cvrChainLoop	`0x2000`	Chain loop: one of the CA certificates recursively signs itself
cvrWeakAlgorithm	`0x4000`	A weak algorithm is used in one of certificates or revocation elements
cvrUserEnforced	`0x8000`	The chain was considered invalid following intervention from a user code

ChainValidationResult

ChainValidities (read-only)

Default: 0

The outcome of a certificate chain validation routine.

Available options:

cvtValid	`0`	The chain is valid
cvtValidButUntrusted	`1`	The chain is valid, but the root certificate is not trusted
cvtInvalid	`2`	The chain is not valid (some of certificates are revoked, expired, or contain an invalid signature)
cvtCantBeEstablished	`3`	The validity of the chain cannot be established because of missing or unavailable validation information (certificates, CRLs, or OCSP responses)

Use the ValidationLog property to access the detailed validation log.

ContainsLongTermInfo

bool (read-only)

Default: false

Returns true if the signature was found to contain long-term validation details (certificates, CRLs, and OCSP response).

EntityLabel

string (read-only)
Default: ""

Use this property to get the timestamp entity label.

This property returns a string label that uniquely identifies the timestamp. The label can be used to establish the signature target in the SignatureFound event or to select the signing chain via the SelectInfo method.

HashAlgorithm

string (read-only)
Default: ""

Returns the timestamp's hash algorithm.

SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_MD2	`MD2`
SB_HASH_ALGORITHM_MD4	`MD4`
SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_CRC32	`CRC32`
SB_HASH_ALGORITHM_SSL3	`SSL3`

SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_POLY1305	`POLY1305`
SB_HASH_ALGORITHM_SHA3_224	`SHA3_224`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`
SB_HASH_ALGORITHM_BLAKE2S_128	`BLAKE2S_128`
SB_HASH_ALGORITHM_BLAKE2S_160	`BLAKE2S_160`
SB_HASH_ALGORITHM_BLAKE2S_224	`BLAKE2S_224`
SB_HASH_ALGORITHM_BLAKE2S_256	`BLAKE2S_256`
SB_HASH_ALGORITHM_BLAKE2B_160	`BLAKE2B_160`
SB_HASH_ALGORITHM_BLAKE2B_256	`BLAKE2B_256`
SB_HASH_ALGORITHM_BLAKE2B_384	`BLAKE2B_384`
SB_HASH_ALGORITHM_BLAKE2B_512	`BLAKE2B_512`
SB_HASH_ALGORITHM_SHAKE_128	`SHAKE_128`
SB_HASH_ALGORITHM_SHAKE_256	`SHAKE_256`
SB_HASH_ALGORITHM_SHAKE_128_LEN	`SHAKE_128_LEN`
SB_HASH_ALGORITHM_SHAKE_256_LEN	`SHAKE_256_LEN`

ParentEntity
string (read-only)

Default: ""

Use this property to get the label of the timestamp's parent entity.

This property references the EntityLabel of the object that the timestamp covers, typically a signature.

SerialNumber

byte[] (read-only)
Default: ""

Returns the timestamp's serial number.

Time

string (read-only)
Default: ""

The time point incorporated into the timestamp.

TimestampType

int (read-only)
Default: 0

Returns the type of the timestamp.

Available options:

tstUnknown	`0`	
tstLegacy	`1`	Supported by: Authenticode components
tstTrusted	`2`	Supported by: Authenticode components
tstGeneric	`3`	Supported by: CAdES components
tstESC	`4`	Supported by: CAdES components
tstContent	`5`	Supported by: CAdES components
tstCertsAndCRLs	`6`	Supported by: CAdES components

tstArchive	7	Archive timestamp. Supported by: ASiC, CAdES, JAdES, Office, SOAP, XAdES components
tstArchive2	8	Archive v2 timestamp. Supported by: ASiC, CAdES components
tstArchive3	9	Archive v3 timestamp. Supported by: ASiC, CAdES components
tstIndividualDataObjects	10	Individual data objects timetamp. Supported by: ASiC, Office, SOAP, XAdES components
tstAllDataObjects	11	All data objects timestamp. Supported by: ASiC, Office, SOAP, XAdES components
tstSignature	12	Signature timestamp. Supported by: ASiC, JAdES, Office, SOAP, XAdES components
tstRefsOnly	13	RefsOnly timestamp. Supported by: ASiC, JAdES, Office, SOAP, XAdES components
tstSigAndRefs	14	SigAndRefs timestamp. Supported by: ASiC, JAdES, Office, SOAP, XAdES components
tstSignedData	15	SignedData timestamp. Supported by: JAdES components
tstArchive141	16	Archive timestamp v1.4.1. Supported by: ASiC, Office, SOAP, XAdES components

Not all of the above timestamp types can be supported by a specific signature technology used (CAdES, PDF, XAdES).

TSAName

string (read-only)

Default: ""

This value uniquely identifies the Timestamp Authority (TSA).

This property provides information about the entity that manages the TSA.

ValidationLog

string (read-only)

Default: ""

Contains the TSA certificate chain validation log. This information is extremely useful if the timestamp validation fails.

ValidationResult

SignatureValidities (read-only)

Default: 0

Contains the timestamp validation outcome.

Use this property to check the result of the most recent timestamp validation.

svtValid	`0`	The signature is valid
svtUnknown	`1`	Signature validity is unknown
svtCorrupted	`2`	The signature is corrupted
svtSignerNotFound	`3`	Failed to acquire the signing certificate. The signature cannot be validated.
svtFailure	`4`	General failure
svtReferenceCorrupted	`5`	Reference corrupted (XML-based signatures only)

Constructors

C# VB.NET

```
public TimestampInfo();
```

Creates a new TimestampInfo object with default field values.

TLSSettings Type

A container for TLS connection settings.

Remarks

The TLS (Transport Layer Security) protocol provides security for information exchanged over insecure connections such as TCP/IP.

The following fields are available:

- AutoValidateCertificates
- BaseConfiguration
- Ciphersuites
- ClientAuth
- ECCurves
- Extensions
- ForceResumeIfDestinationChanges
- PreSharedIdentity
- PreSharedKey
- PreSharedKeyCiphersuite
- RenegotiationAttackPreventionMode
- RevocationCheck
- SSLOptions
- TLSMode
- UseExtendedMasterSecret
- UseSessionResumption
- Versions

Fields

AutoValidateCertificates
bool
Default: true

Specifies whether server-side TLS certificates should be validated automatically using internal validation rules.

BaseConfiguration
SecureTransportPredefinedConfigurations
Default: 0

Selects the base configuration for the TLS settings. Several profiles are offered and tuned up for different purposes, such as high security or higher compatibility.

stpcDefault	`0`
stpcCompatible	`1`
stpcComprehensiveInsecure	`2`
stpcHighlySecure	`3`

Ciphersuites

string

Default: ""

A list of ciphersuites separated with commas or semicolons. Each ciphersuite in the list may be prefixed with a minus sign (-) to indicate that the ciphersuite should be disabled rather than enabled. Besides the specific ciphersuite modifiers, this property supports the *all* (and *-all*) aliases, allowing all ciphersuites to be blanketly enabled or disabled at once.

Note: the list of ciphersuites provided to this property alters the baseline list of ciphersuites as defined by BaseConfiguration. Remember to start your ciphersuite string with -all; if you need to only enable a specific fixed set of ciphersuites. The list of supported ciphersuites is provided below:

- NULL_NULL_NULL
- RSA_NULL_MD5
- RSA_NULL_SHA
- RSA_RC4_MD5
- RSA_RC4_SHA
- RSA_RC2_MD5
- RSA_IDEA_MD5
- RSA_IDEA_SHA
- RSA_DES_MD5
- RSA_DES_SHA
- RSA_3DES_MD5
- RSA_3DES_SHA
- RSA_AES128_SHA
- RSA_AES256_SHA
- DH_DSS_DES_SHA
- DH_DSS_3DES_SHA
- DH_DSS_AES128_SHA

- DH_DSS_AES256_SHA
- DH_RSA_DES_SHA
- DH_RSA_3DES_SHA
- DH_RSA_AES128_SHA
- DH_RSA_AES256_SHA
- DHE_DSS_DES_SHA
- DHE_DSS_3DES_SHA
- DHE_DSS_AES128_SHA
- DHE_DSS_AES256_SHA
- DHE_RSA_DES_SHA
- DHE_RSA_3DES_SHA
- DHE_RSA_AES128_SHA
- DHE_RSA_AES256_SHA
- DH_ANON_RC4_MD5
- DH_ANON_DES_SHA
- DH_ANON_3DES_SHA
- DH_ANON_AES128_SHA
- DH_ANON_AES256_SHA
- RSA_RC2_MD5_EXPORT
- RSA_RC4_MD5_EXPORT
- RSA_DES_SHA_EXPORT
- DH_DSS_DES_SHA_EXPORT
- DH_RSA_DES_SHA_EXPORT
- DHE_DSS_DES_SHA_EXPORT
- DHE_RSA_DES_SHA_EXPORT
- DH_ANON_RC4_MD5_EXPORT
- DH_ANON_DES_SHA_EXPORT
- RSA_CAMELLIA128_SHA
- DH_DSS_CAMELLIA128_SHA
- DH_RSA_CAMELLIA128_SHA
- DHE_DSS_CAMELLIA128_SHA
- DHE_RSA_CAMELLIA128_SHA
- DH_ANON_CAMELLIA128_SHA
- RSA_CAMELLIA256_SHA
- DH_DSS_CAMELLIA256_SHA
- DH_RSA_CAMELLIA256_SHA
- DHE_DSS_CAMELLIA256_SHA
- DHE_RSA_CAMELLIA256_SHA
- DH_ANON_CAMELLIA256_SHA
- PSK_RC4_SHA
- PSK_3DES_SHA
- PSK_AES128_SHA
- PSK_AES256_SHA

- DHE_PSK_RC4_SHA
- DHE_PSK_3DES_SHA
- DHE_PSK_AES128_SHA
- DHE_PSK_AES256_SHA
- RSA_PSK_RC4_SHA
- RSA_PSK_3DES_SHA
- RSA_PSK_AES128_SHA
- RSA_PSK_AES256_SHA
- RSA_SEED_SHA
- DH_DSS_SEED_SHA
- DH_RSA_SEED_SHA
- DHE_DSS_SEED_SHA
- DHE_RSA_SEED_SHA
- DH_ANON_SEED_SHA
- SRP_SHA_3DES_SHA
- SRP_SHA_RSA_3DES_SHA
- SRP_SHA_DSS_3DES_SHA
- SRP_SHA_AES128_SHA
- SRP_SHA_RSA_AES128_SHA
- SRP_SHA_DSS_AES128_SHA
- SRP_SHA_AES256_SHA
- SRP_SHA_RSA_AES256_SHA
- SRP_SHA_DSS_AES256_SHA
- ECDH_ECDSA_NULL_SHA
- ECDH_ECDSA_RC4_SHA
- ECDH_ECDSA_3DES_SHA
- ECDH_ECDSA_AES128_SHA
- ECDH_ECDSA_AES256_SHA
- ECDHE_ECDSA_NULL_SHA
- ECDHE_ECDSA_RC4_SHA
- ECDHE_ECDSA_3DES_SHA
- ECDHE_ECDSA_AES128_SHA
- ECDHE_ECDSA_AES256_SHA
- ECDH_RSA_NULL_SHA
- ECDH_RSA_RC4_SHA
- ECDH_RSA_3DES_SHA
- ECDH_RSA_AES128_SHA
- ECDH_RSA_AES256_SHA
- ECDHE_RSA_NULL_SHA
- ECDHE_RSA_RC4_SHA
- ECDHE_RSA_3DES_SHA
- ECDHE_RSA_AES128_SHA
- ECDHE_RSA_AES256_SHA

- ECDH_ANON_NULL_SHA
- ECDH_ANON_RC4_SHA
- ECDH_ANON_3DES_SHA
- ECDH_ANON_AES128_SHA
- ECDH_ANON_AES256_SHA
- RSA_NULL_SHA256
- RSA_AES128_SHA256
- RSA_AES256_SHA256
- DH_DSS_AES128_SHA256
- DH_RSA_AES128_SHA256
- DHE_DSS_AES128_SHA256
- DHE_RSA_AES128_SHA256
- DH_DSS_AES256_SHA256
- DH_RSA_AES256_SHA256
- DHE_DSS_AES256_SHA256
- DHE_RSA_AES256_SHA256
- DH_ANON_AES128_SHA256
- DH_ANON_AES256_SHA256
- RSA_AES128_GCM_SHA256
- RSA_AES256_GCM_SHA384
- DHE_RSA_AES128_GCM_SHA256
- DHE_RSA_AES256_GCM_SHA384
- DH_RSA_AES128_GCM_SHA256
- DH_RSA_AES256_GCM_SHA384
- DHE_DSS_AES128_GCM_SHA256
- DHE_DSS_AES256_GCM_SHA384
- DH_DSS_AES128_GCM_SHA256
- DH_DSS_AES256_GCM_SHA384
- DH_ANON_AES128_GCM_SHA256
- DH_ANON_AES256_GCM_SHA384
- ECDHE_ECDSA_AES128_SHA256
- ECDHE_ECDSA_AES256_SHA384
- ECDH_ECDSA_AES128_SHA256
- ECDH_ECDSA_AES256_SHA384
- ECDHE_RSA_AES128_SHA256
- ECDHE_RSA_AES256_SHA384
- ECDH_RSA_AES128_SHA256
- ECDH_RSA_AES256_SHA384
- ECDHE_ECDSA_AES128_GCM_SHA256
- ECDHE_ECDSA_AES256_GCM_SHA384
- ECDH_ECDSA_AES128_GCM_SHA256
- ECDH_ECDSA_AES256_GCM_SHA384
- ECDHE_RSA_AES128_GCM_SHA256

- ECDHE_RSA_AES256_GCM_SHA384
- ECDH_RSA_AES128_GCM_SHA256
- ECDH_RSA_AES256_GCM_SHA384
- PSK_AES128_GCM_SHA256
- PSK_AES256_GCM_SHA384
- DHE_PSK_AES128_GCM_SHA256
- DHE_PSK_AES256_GCM_SHA384
- RSA_PSK_AES128_GCM_SHA256
- RSA_PSK_AES256_GCM_SHA384
- PSK_AES128_SHA256
- PSK_AES256_SHA384
- PSK_NULL_SHA256
- PSK_NULL_SHA384
- DHE_PSK_AES128_SHA256
- DHE_PSK_AES256_SHA384
- DHE_PSK_NULL_SHA256
- DHE_PSK_NULL_SHA384
- RSA_PSK_AES128_SHA256
- RSA_PSK_AES256_SHA384
- RSA_PSK_NULL_SHA256
- RSA_PSK_NULL_SHA384
- RSA_CAMELLIA128_SHA256
- DH_DSS_CAMELLIA128_SHA256
- DH_RSA_CAMELLIA128_SHA256
- DHE_DSS_CAMELLIA128_SHA256
- DHE_RSA_CAMELLIA128_SHA256
- DH_ANON_CAMELLIA128_SHA256
- RSA_CAMELLIA256_SHA256
- DH_DSS_CAMELLIA256_SHA256
- DH_RSA_CAMELLIA256_SHA256
- DHE_DSS_CAMELLIA256_SHA256
- DHE_RSA_CAMELLIA256_SHA256
- DH_ANON_CAMELLIA256_SHA256
- ECDHE_ECDSA_CAMELLIA128_SHA256
- ECDHE_ECDSA_CAMELLIA256_SHA384
- ECDH_ECDSA_CAMELLIA128_SHA256
- ECDH_ECDSA_CAMELLIA256_SHA384
- ECDHE_RSA_CAMELLIA128_SHA256
- ECDHE_RSA_CAMELLIA256_SHA384
- ECDH_RSA_CAMELLIA128_SHA256
- ECDH_RSA_CAMELLIA256_SHA384
- RSA_CAMELLIA128_GCM_SHA256
- RSA_CAMELLIA256_GCM_SHA384

- DHE_RSA_CAMELLIA128_GCM_SHA256
- DHE_RSA_CAMELLIA256_GCM_SHA384
- DH_RSA_CAMELLIA128_GCM_SHA256
- DH_RSA_CAMELLIA256_GCM_SHA384
- DHE_DSS_CAMELLIA128_GCM_SHA256
- DHE_DSS_CAMELLIA256_GCM_SHA384
- DH_DSS_CAMELLIA128_GCM_SHA256
- DH_DSS_CAMELLIA256_GCM_SHA384
- DH_anon_CAMELLIA128_GCM_SHA256
- DH_anon_CAMELLIA256_GCM_SHA384
- ECDHE_ECDSA_CAMELLIA128_GCM_SHA256
- ECDHE_ECDSA_CAMELLIA256_GCM_SHA384
- ECDH_ECDSA_CAMELLIA128_GCM_SHA256
- ECDH_ECDSA_CAMELLIA256_GCM_SHA384
- ECDHE_RSA_CAMELLIA128_GCM_SHA256
- ECDHE_RSA_CAMELLIA256_GCM_SHA384
- ECDH_RSA_CAMELLIA128_GCM_SHA256
- ECDH_RSA_CAMELLIA256_GCM_SHA384
- PSK_CAMELLIA128_GCM_SHA256
- PSK_CAMELLIA256_GCM_SHA384
- DHE_PSK_CAMELLIA128_GCM_SHA256
- DHE_PSK_CAMELLIA256_GCM_SHA384
- RSA_PSK_CAMELLIA128_GCM_SHA256
- RSA_PSK_CAMELLIA256_GCM_SHA384
- PSK_CAMELLIA128_SHA256
- PSK_CAMELLIA256_SHA384
- DHE_PSK_CAMELLIA128_SHA256
- DHE_PSK_CAMELLIA256_SHA384
- RSA_PSK_CAMELLIA128_SHA256
- RSA_PSK_CAMELLIA256_SHA384
- ECDHE_PSK_CAMELLIA128_SHA256
- ECDHE_PSK_CAMELLIA256_SHA384
- ECDHE_PSK_RC4_SHA
- ECDHE_PSK_3DES_SHA
- ECDHE_PSK_AES128_SHA
- ECDHE_PSK_AES256_SHA
- ECDHE_PSK_AES128_SHA256
- ECDHE_PSK_AES256_SHA384
- ECDHE_PSK_NULL_SHA
- ECDHE_PSK_NULL_SHA256
- ECDHE_PSK_NULL_SHA384
- ECDHE_RSA_CHACHA20_POLY1305_SHA256
- ECDHE_ECDSA_CHACHA20_POLY1305_SHA256

- DHE_RSA_CHACHA20_POLY1305_SHA256
- PSK_CHACHA20_POLY1305_SHA256
- ECDHE_PSK_CHACHA20_POLY1305_SHA256
- DHE_PSK_CHACHA20_POLY1305_SHA256
- RSA_PSK_CHACHA20_POLY1305_SHA256
- AES128_GCM_SHA256
- AES256_GCM_SHA384
- CHACHA20_POLY1305_SHA256
- AES128_CCM_SHA256
- AES128_CCM8_SHA256

ClientAuth

ClientAuthTypes

Default: 0

Enables or disables certificate-based client authentication.

Set this property to true to tune up the client authentication type:

ccatNoAuth	`0`
ccatRequestCert	`1`
ccatRequireCert	`2`

ECCurves

string

Default: ""

Defines the elliptic curves to enable.

Extensions

string

Default: ""

Provides access to TLS extensions.

ForceResumeIfDestinationChanges

bool

Default: false

Whether to force TLS session resumption when the destination address changes.

PreSharedIdentity

string

Default: ""

Defines the identity used when the PSK (Pre-Shared Key) key-exchange mechanism is negotiated.

PreSharedKey

string

Default: ""

Contains the pre-shared key for the PSK (Pre-Shared Key) key-exchange mechanism, encoded with base16.

PreSharedKeyCiphersuite

string

Default: ""

Defines the ciphersuite used for PSK (Pre-Shared Key) negotiation.

RenegotiationAttackPreventionMode

RenegotiationAttackPreventionModes

Default: 2

Selects the renegotiation attack prevention mechanism.

The following options are available:

crapmCompatible	`0`	TLS 1.0 and 1.1 compatibility mode (renegotiation indication extension is disabled).
crapmStrict	`1`	Renegotiation attack prevention is enabled and enforced.
crapmAuto	`2`	Automatically choose whether to enable or disable renegotiation attack prevention.

RevocationCheck

RevocationCheckKinds

Default: 1

Specifies the kind(s) of revocation check to perform.

Revocation checking is necessary to ensure the integrity of the chain and obtain up-to-date certificate validity and trustworthiness information.

crcNone	`0`	No revocation checking.
crcAuto	`1`	Automatic mode selection. Currently this maps to crcAnyOCSPOrCRL, but it may change in the future.
crcAllCRL	`2`	All provided CRL endpoints will be checked, and all checks must succeed.
crcAllOCSP	`3`	All provided OCSP endpoints will be checked, and all checks must succeed.
crcAllCRLAndOCSP	`4`	All provided CRL and OCSP endpoints will be checked, and all checks must succeed.
crcAnyCRL	`5`	All provided CRL endpoints will be checked, and at least one check must succeed.
crcAnyOCSP	`6`	All provided OCSP endpoints will be checked, and at least one check must succeed.
crcAnyCRLOrOCSP	`7`	All provided CRL and OCSP endpoints will be checked, and at least one check must succeed. CRL endpoints are checked first.
crcAnyOCSPOrCRL	`8`	All provided CRL and OCSP endpoints will be checked, and at least one check must succeed. OCSP endpoints are checked first.

This setting controls the way the revocation checks are performed for every certificate in the chain. Typically certificates come with two types of revocation information sources: CRL (certificate revocation lists) and OCSP responders. CRLs are static objects periodically published by the CA at some online location. OCSP responders are active online services maintained by the CA that can provide up-to-date information on certificate statuses in near real time.

There are some conceptual differences between the two. CRLs are normally larger in size. Their use involves some latency because there is normally some delay between the time when a certificate was revoked and the time the subsequent CRL mentioning that is published. The benefits of CRL is that the

same object can provide statuses for all certificates issued by a particular CA, and that the whole technology is much simpler than OCSP (and thus is supported by more CAs).

This setting lets you adjust the validation course by including or excluding certain types of revocation sources from the validation process. The crcAnyOCSPOrCRL setting (give preference to the faster OCSP route and only demand one source to succeed) is a good choice for most typical validation environments. The "crcAll*" modes are much stricter, and may be used in scenarios where bulletproof validity information is essential.

NOTE: If no CRL or OCSP endpoints are provided by the CA, the revocation check will be considered successful. This is because the CA chose not to supply revocation information for its certificates, meaning they are considered irrevocable.

NOTE: Within each of the above settings, if any retrieved CRL or OCSP response indicates that the certificate has been revoked, the revocation check fails.

SSLOptions

int

Default: 16

Various SSL (TLS) protocol options, set of

cssloExpectShutdownMessage	`0x001`	Wait for the close-notify message when shutting down the connection
cssloOpenSSLDTLSWorkaround	`0x002`	(DEPRECATED) Use a DTLS version workaround when talking to very old OpenSSL versions
cssloDisableKexLengthAlignment	`0x004`	Do not align the client-side PMS by the RSA modulus size. It is unlikely that you will ever need to adjust it.
cssloForceUseOfClientCertHashAlg	`0x008`	Enforce the use of the client certificate hash algorithm. It is unlikely that you will ever need to adjust it.
cssloAutoAddServerNameExtension	`0x010`	Automatically add the server name extension when known

cssloAcceptTrustedSRPPrimesOnly	`0x020`	Accept trusted SRP primes only
cssloDisableSignatureAlgorithmsExtension	`0x040`	Disable (do not send) the signature algorithms extension. It is unlikely that you will ever need to adjust it.
cssloIntolerateHigherProtocolVersions	`0x080`	(server option) Do not allow fallback from TLS versions higher than currently enabled
cssloStickToPrefCertHashAlg	`0x100`	Stick to preferred certificate hash algorithms
cssloNoImplicitTLS12Fallback	`0x200`	Disable implicit TLS 1.3 to 1.2 fallbacks
cssloUseHandshakeBatches	`0x400`	Send the handshake message as large batches rather than individually

TLSMode

SSLModes

Default: 0

Specifies the TLS mode to use.

smDefault	`0`	
smNoTLS	`1`	Do not use TLS
smExplicitTLS	`2`	Connect to the server without any encryption and then request an SSL session.
smImplicitTLS	`3`	Connect to the specified port, and establish the SSL session at once.
smMixedTLS	`4`	Connect to the specified port, and establish the SSL session at once, but allow plain data.

UseExtendedMasterSecret

bool

Default: false

Enables the Extended Master Secret Extension, as defined in RFC 7627.

UseSessionResumption

bool

Default: false

Enables or disables the TLS session resumption capability.

Versions

int

Default: 16

The SSL/TLS versions to enable by default.

csbSSL2	`0x01`	SSL 2
csbSSL3	`0x02`	SSL 3
csbTLS1	`0x04`	TLS 1.0
csbTLS11	`0x08`	TLS 1.1
csbTLS12	`0x10`	TLS 1.2
csbTLS13	`0x20`	TLS 1.3

Constructors

C# VB.NET

```
public TLSSettings();
```

Creates a new TLSSettings object.

XAdESSignature Type

The component is a container for an XAdES/XML signature.

Remarks

XML document may include any number of XAdES/XML signatures. component stores on of them.

The following fields are available:

- CanonicalizationMethod
- ChainValidationDetails
- ChainValidationResult
- ClaimedSigningTime
- CompatibilityErrors
- ContainsLongTermInfo
- EntityLabel
- HashAlgorithm
- IssuerRDN
- LastArchivalTime
- Level
- ParentEntity
- PolicyHash
- PolicyHashAlgorithm
- PolicyID
- PolicyURI
- SerialNumber
- SignatureBytes
- SignatureType
- SignatureValidationResult
- SubjectKeyID
- SubjectRDN
- Timestamped
- ValidatedSigningTime
- ValidationLog
- XAdESVersion
- XMLElement

Fields

CanonicalizationMethod
XMLCanonicalizationMethods

Default: 0

The XML canonicalization method that was used for signing.

Supported canonicalization methods:

cxcmNone	`0`
cxcmCanon	`1`
cxcmCanonComment	`2`
cxcmExclCanon	`3`
cxcmExclCanonComment	`4`
cxcmMinCanon	`5`
cxcmCanon_v1_1	`6`
cxcmCanonComment_v1_1	`7`

ChainValidationDetails

int (read-only)

Default: 0

The details of a certificate chain validation outcome. They may often suggest the reasons that contributed to the overall validation result.

Returns a bit mask of the following options:

cvrBadData	`0x0001`	One or more certificates in the validation path are malformed
cvrRevoked	`0x0002`	One or more certificates are revoked
cvrNotYetValid	`0x0004`	One or more certificates are not yet valid
cvrExpired	`0x0008`	One or more certificates are expired

cvrInvalidSignature	`0x0010`	A certificate contains a non-valid digital signature
cvrUnknownCA	`0x0020`	A CA certificate for one or more certificates has not been found (chain incomplete)
cvrCAUnauthorized	`0x0040`	One of the CA certificates are not authorized to act as CA
cvrCRLNotVerified	`0x0080`	One or more CRLs could not be verified
cvrOCSPNotVerified	`0x0100`	One or more OCSP responses could not be verified
cvrIdentityMismatch	`0x0200`	The identity protected by the certificate (a TLS endpoint or an e-mail addressee) does not match what is recorded in the certificate
cvrNoKeyUsage	`0x0400`	A mandatory key usage is not enabled in one of the chain certificates
cvrBlocked	`0x0800`	One or more certificates are blocked
cvrFailure	`0x1000`	General validation failure
cvrChainLoop	`0x2000`	Chain loop: one of the CA certificates recursively signs itself
cvrWeakAlgorithm	`0x4000`	A weak algorithm is used in one of certificates or revocation elements
cvrUserEnforced	`0x8000`	The chain was considered invalid following intervention from a user code

ChainValidationResult

ChainValidities (read-only)

Default: 0

The outcome of a certificate chain validation routine.

Available options:

cvtValid	`0`	The chain is valid
cvtValidButUntrusted	`1`	The chain is valid, but the root certificate is not trusted
cvtInvalid	`2`	The chain is not valid (some of certificates are revoked, expired, or contain an invalid signature)
cvtCantBeEstablished	`3`	The validity of the chain cannot be established because of missing or unavailable validation information (certificates, CRLs, or OCSP responses)

Use the ValidationLog property to access the detailed validation log.

ClaimedSigningTime

string
Default: ""

The signing time from the signer's computer.

Use this property to provide the signature production time. The claimed time is not supported by a trusted source; it may be inaccurate, forfeited, or wrong, and as such is usually taken for informational purposes only by verifiers. Use timestamp servers to embed verifiable trusted timestamps. The time is in UTC.

CompatibilityErrors

int (read-only)
Default: 0

Returns compatibility errors encountered during validation.

Use this property to get specific compatibility errors encountered during validation. Unlike chain validation details, compatibility errors indicate violations by the signature of the assumed signature level/profile. For example, BES signatures are required to contain the signing time attribute. A prospective BES signature without such attribute will invoke a compatibility error.

Supported values:

xceUnknown	`0x00001`	Unknown validation error
xceInconsistentSigningCertificate	`0x00010`	Inconsistent signing certificate

ContainsLongTermInfo

bool (read-only)
Default: false

Returns true if the signature was found to contain long-term validation details (certificates, CRLs, and OCSP response).

EntityLabel

string (read-only)
Default: ""

Use this property to get the signature entity label.

This property returns a string label that uniquely identifies the signature. The label can be used to establish the signature target in the SignatureFound event or to select the signing chain via the SelectInfo method.

HashAlgorithm

string
Default: "unknown"

The hash algorithm used for signing.

SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`

SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`

IssuerRDN

string (read-only)
Default: ""

The Relative Distinguished Name of the signing certificate's issuer.

A collection of information, in the form of [OID, Value] pairs, about the company that issued the signing certificate.

LastArchivalTime

string (read-only)
Default: ""

Indicates the most recent archival time of an archived signature

This property returns the time of the most recent archival timestamp applied to the signature. This property only makes sense for 'archived' (e.g. CAdES-A) signatures. Time is in UTC.

Level

AdESSignatureLevels
Default: 2

Specifies which level or form of XAdES should be produced.

Use this property to specify the level or form of advanced electronic signature to be produced.

Use the *Generic* value to produce XML-DSIG signature.

The supported levels and forms are:

aslUnknown	0	Unknown signature level
aslGeneric	1	Generic (this value applicable to XAdES signature only and corresponds to XML-DSIG signature)
aslBaselineB	2	Baseline B (B-B, basic)
aslBaselineT	3	Baseline T (B-T, timestamped)
aslBaselineLT	4	Baseline LT (B-LT, long-term)
aslBaselineLTA	5	Baseline LTA (B-LTA, long-term with archived timestamp)
aslBES	6	BES (Basic Electronic Signature)
aslEPES	7	EPES (Electronic Signature with an Explicit Policy)
aslT	8	T (Timestamped)
aslC	9	C (T with revocation references)
aslX	10	X (C with SigAndRefs timestamp or RefsOnly timestamp) (this value applicable to XAdES signature only)
aslXType1	11	X Type 1 (C with an ES-C timestamp) (this value applicable to CAdES signature only)
aslXType2	12	X Type 2 (C with a CertsAndCRLs timestamp) (this value applicable to CAdES signature only)
aslXL	13	X-L (X with revocation values) (this value applicable to XAdES signature only)
aslXLType1	14	X-L Type 1 (C with revocation values and an ES-C timestamp) (this value applicable to CAdES signature only)

aslXLType2	15	X-L Type 2 (C with revocation values and a CertsAndCRLs timestamp) (this value applicable to CAdES signature only)
aslA	16	A (archived)
aslExtendedBES	17	Extended BES
aslExtendedEPES	18	Extended EPES
aslExtendedT	19	Extended T
aslExtendedC	20	Extended C
aslExtendedX	21	Extended X (this value applicable to XAdES signature only)
aslExtendedXType1	22	Extended X (type 1) (this value applicable to CAdES signature only)
aslExtendedXType2	23	Extended X (type 2) (this value applicable to CAdES signature only)
aslExtendedXLong	24	Extended X-Long (this value applicable to XAdES signature only)
aslExtendedXL	25	Extended X-L (this value applicable to XAdES signature only)
aslExtendedXLType1	26	Extended XL (type 1) (this value applicable to CAdES signature only)
aslExtendedXLType2	27	Extended XL (type 2) (this value applicable to CAdES signature only)
aslExtendedA	28	Extended A

* For XAdES form from XAdES v1.1.1 use either BES or EPES form values ** Extended forms are supported starting from XAdES v1.3.2

ParentEntity
string
Default: ""

Use this property to get the parent signature label.

This property contains the unique entity label of the current signature's parent object - typically a higher-level signature or a timestamp.

PolicyHash

string
Default: ""

The signature policy hash value.

Use this property to get the signature policy hash from EPES signatures

---

PolicyHashAlgorithm

string
Default: ""

The algorithm that was used to calculate the signature policy hash

Use this property to get or set the hash algorithm used to calculate the signature policy hash. Read the actual hash value from PolicyHash.

SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_MD2	`MD2`
SB_HASH_ALGORITHM_MD4	`MD4`
SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_CRC32	`CRC32`

SB_HASH_ALGORITHM_SSL3	`SSL3`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_POLY1305	`POLY1305`
SB_HASH_ALGORITHM_SHA3_224	`SHA3_224`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`
SB_HASH_ALGORITHM_BLAKE2S_128	`BLAKE2S_128`
SB_HASH_ALGORITHM_BLAKE2S_160	`BLAKE2S_160`
SB_HASH_ALGORITHM_BLAKE2S_224	`BLAKE2S_224`
SB_HASH_ALGORITHM_BLAKE2S_256	`BLAKE2S_256`
SB_HASH_ALGORITHM_BLAKE2B_160	`BLAKE2B_160`
SB_HASH_ALGORITHM_BLAKE2B_256	`BLAKE2B_256`
SB_HASH_ALGORITHM_BLAKE2B_384	`BLAKE2B_384`
SB_HASH_ALGORITHM_BLAKE2B_512	`BLAKE2B_512`
SB_HASH_ALGORITHM_SHAKE_128	`SHAKE_128`
SB_HASH_ALGORITHM_SHAKE_256	`SHAKE_256`
SB_HASH_ALGORITHM_SHAKE_128_LEN	`SHAKE_128_LEN`
SB_HASH_ALGORITHM_SHAKE_256_LEN	`SHAKE_256_LEN`

PolicyID

string

Default: ""

The policy ID that was included or to be included into the signature.

Use this property to retrieve the signature policy identifier from EPES signatures.

PolicyURI

string

Default: ""

The signature policy URI that was included in the signature.

Use this property to set or retrieve the URI of the signature policy from EPES signatures.

SerialNumber

byte[] (read-only)

Default: ""

The serial number of the signing certificate.

SignatureBytes

byte[] (read-only)

Default: ""

Returns the binary representation of the XML-DSig/XAdES signature.

SignatureType

XMLSignatureTypes

Default: 4

The signature type to employ when signing the document.

This property specifies the signature type to be used when signing the document.

Supported values:

cxstDetached	1	Specifies whether a detached signature should be produced. I.e., a signature which is kept separately from the signed document.
cxstEnveloping	2	Specifies whether an enveloping signature should be produced.

cxstEnveloped	4	Specifies whether an enveloped signature should be produced.

SignatureValidationResult

SignatureValidities (read-only)

Default: 0

The outcome of the cryptographic signature validation.

The following signature validity values are supported:

svtValid	0	The signature is valid
svtUnknown	1	Signature validity is unknown
svtCorrupted	2	The signature is corrupted
svtSignerNotFound	3	Failed to acquire the signing certificate. The signature cannot be validated.
svtFailure	4	General failure
svtReferenceCorrupted	5	Reference corrupted (XML-based signatures only)

SubjectKeyID

byte[] (read-only)

Default: ""

Contains the subject key identifier of the signing certificate.

Subject Key Identifier is a (non-critical) X.509 certificate extension which allows the identification of certificates containing a particular public key. In SecureBlackbox, the unique identifier is represented by a SHA-1 hash of the bit string of the subject public key.

SubjectRDN

string (read-only)

Default: ""

Contains information about the person owning the signing certificate. Only certificates with given subject information will be enumerated during the search operation. Information is stored in the form of [Object Identifier, Value] pairs.

Timestamped

bool (read-only)
Default: false

Use this property to establish whether the signature contains an embedded timestamp.

ValidatedSigningTime

string (read-only)
Default: ""

Contains the certified signing time.

Use this property to obtain the signing time as certified by a timestamp from a trusted timestamping authority. This property is only non-empty if there was a valid timestamp included in the signature.

ClaimedSigningTime returns a non-trusted signing time from the signer's computer.

Both times are in UTC.

ValidationLog

string (read-only)
Default: ""

Contains the complete log of the certificate validation routine.

Use this property to access the chain validation log produced by the component. The log can be very useful when investigating issues with chain validation, as it contains a step-by-step trace of the entire validation procedure.

XAdESVersion

XAdESVersions
Default: 3

Specifies XAdES version.

This property specifies the version of the XAdES specification the signature should comply with.

The supported vesions are:

xavUnknown	`0`	Unknown
xav111	`1`	XAdES v1.1.1
xav122	`2`	XAdES v1.2.2
xav132	`3`	XAdES v1.3.2
xav141	`4`	XAdES v1.4.1 (aka v1.4.2)

XMLElement

string

Default: ""

Specifies the XML element where to save the signature or containing the signature.

This property specifies the XML element where to save the electronic signature or that contains the signature to be validated.

Supported values are:

""	an empty string indicates the Document element
"#id"	indicates an XML element with specified Id
XPath expression	indicates an XML element selected using XPath expression. Use AddKnownNamespace method to specify Prefixes and NamespaceURIs For example: "/root/data[1]" - indicates the second "data" element under the document element with a name "root" "//ns1:data" - indicates a data element. "ns1" prefix should be defined via AddKnownNamespace method.
Node name	indicates an XML element selected using its NodeName. For example: "data" - indicates an XML element with node name "data".

Constructors

C# VB.NET

```
public XAdESSignature();
```

Creates a new XAdES signature object.

XMLReference Type

Represents an XML reference element.

Remarks

XMLReference specifies the digest algorithm and digest value, and, optionally: an identifier of the object being signed, the type of the object, and/or a list of transforms to be applied prior to digesting.

The following fields are available:

- AutoGenerateElementId
- CanonicalizationMethod
- CustomElementId
- DataObjectDescription
- DataObjectEncoding
- DataObjectIdentifier
- DataObjectMimeType
- DigestValue
- HasDataObjectFormat
- HashAlgorithm
- HasURI
- ID
- InclusiveNamespacesPrefixList
- ReferenceType
- TargetData
- TargetType
- TargetXMLElement
- URI
- UseBase64Transform
- UseEnvelopedSignatureTransform
- UseXPathFilter2Transform
- UseXPathTransform

- ValidationResult
- XPathExpression
- XPathFilter2Expressions
- XPathFilter2Filters
- XPathFilter2PrefixList
- XPathPrefixList

Fields

AutoGenerateElementId

bool

Default: false

Specifies whether the identifier (ID) attribute for a referenced (target) element should be auto-generated during signing. Used when the referenced element doesn't have an ID and *CustomElementId* and *URI* properties are empty.

CanonicalizationMethod

XMLCanonicalizationMethods

Default: 0

Use this property to specify the canonicalization method for the transform of the reference. Use cxcmNone value to not to include canonicalization transform in transform chain. See XML-Signature Syntax and Processing specification for details.

cxcmNone	`0`
cxcmCanon	`1`
cxcmCanonComment	`2`
cxcmExclCanon	`3`
cxcmExclCanonComment	`4`
cxcmMinCanon	`5`
cxcmCanon_v1_1	`6`
cxcmCanonComment_v1_1	`7`

CustomElementId

string

Default: ""

Specifies a custom identifier (ID) attribute for a referenced (target) element that will be set on signing. Used when the referenced element doesn't have an ID and *URI* property is empty.

DataObjectDescription

string

Default: ""

This property contains textual information related to the referenced data object, which is found in the corresponding DataObjectFormat's Description element.

DataObjectEncoding

string

Default: ""

This property contains an indication of the encoding format of the referenced data object, which is found in the corresponding DataObjectFormat's Encoding element.

DataObjectIdentifier

string

Default: ""

This property contains an identifier indicating the type of the referenced data object, which is found in the corresponding DataObjectFormat's ObjectIdentifier element.

DataObjectMimeType

string

Default: ""

This property contains an indication of the MIME type of the referenced data object, which is found in the corresponding DataObjectFormat's MimeType element.

DigestValue

byte[]

Default: ""

Use this property to get or set the value of the digest calculated over the referenced data.

This field is optional and should be set only if you don't provide the actual data via *TargetData* or *URI*. If the data is set, then you don't need to set *DigestValue* since it will be calculated automatically.

HasDataObjectFormat

bool

Default: false

Specifies whether corresponding XAdES DataObjectFormat element is created for the current reference.

Check this property to find out if the reference has an associated DataObjectFormat element.

HashAlgorithm

string

Default: "sha256"

Specifies the hash algorithm to be used.

Supported values:

SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`

SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`

HasURI
bool
Default: true

Specifies whether the *URI* is set (even when it is empty).

ID
string
Default: ""

A user-defined identifier (ID) attribute of this Reference element.

InclusiveNamespacesPrefixList
string
Default: ""

Use this property to specify InclusiveNamespaces PrefixList for exclusive canonicalization transform of the reference. See XML-Signature Syntax and Processing specification for details.

ReferenceType
string
Default: ""

The Reference's *type* attribute as defined in XMLDSIG specification.

TargetData
byte[]
Default: ""

Contains the referenced external data when the digest value is not explicitly specified.

This field is optional and should only be set if you reference the external data via *URI*, and you don't provide the digest value explicitly via *DigestValue*.

TargetType
XMLReferenceTargetTypes

Default: 0

The reference's target type to use.

Use this property to specify the reference's target type to use when forming the signature.

TargetXMLElement

string
Default: ""

This property specifies the referenced XML element. Used when the *URI* property is not set. In this case, the URI value is generated based on the ID of the referenced (target) XML element. If the *URI* property is set, this property is ignored until the *ResolveReference* event.

Supported values are:

""	an empty string indicates the Document element.
"#id"	indicates an XML element with specified Id.
XPointer expression	indicates an XML element selected using XPointer expression. Use the AddKnownNamespace method to specify Prefixes and NamespaceURIs For example: "/root/data[1]" - indicates the second "data" element under the document element with a name "root" "//ns1:data" - indicates a data element. "ns1" prefix should be defined via AddKnownNamespace method.
Node name	indicates an XML element selected using its NodeName. For example: "data" - indicates an XML element with node name "data".

URI

string
Default: ""

Use this property to get or set the URL which references the data. If the data is external, the application must set either *TargetData* or *DigestValue*. If *TargetData* is set, the digest is calculated automatically unless it is explicitly set by the application via *DigestValue*.

UseBase64Transform

bool

Default: false

Specifies whether Base64 transform is included in transform chain.

UseEnvelopedSignatureTransform

bool

Default: false

Specifies whether enveloped signature transform is included in transform chain.

UseXPathFilter2Transform

bool

Default: false

Specifies whether XPath Filter 2.0 transform is included in transform chain.

UseXPathTransform

bool

Default: false

Specifies whether XPath transform is included in transform chain.

ValidationResult

bool (read-only)

Default: false

The outcome of the cryptographic reference validation.

XPathExpression

string

Default: ""

Use this property to specify XPath expression for XPath transform of the reference.

XPathFilter2Expressions

string

Default: ""

Use this property to specify XPointer expression(s) for XPath Filter 2.0 transform of the reference.

XPathFilter2Filters

string

Default: ""

Use this property to specify XPointer filter(s) for XPath Filter 2.0 transform of the reference. The prefix list is comma-separated.

Supported values:

"intersect"	Intersect filter computes the intersection of the selected subtrees with the filter node-set.
"subtract"	Subtract filter computes the subtraction of the selected subtrees with the filter node-set.
"union"	Union filter computes the union of the selected subtrees with the filter node-set.

XPathFilter2PrefixList

string

Default: ""

Use this property to specify a prefix list for XPath Filter 2.0 transform of the reference. The prefix list is space-separated. Namespace URIs that are used are taken from *XPathNamespaces* property.

XPathPrefixList

string

Default: ""

Use this property to specify a prefix list for XPath transform of the reference. The prefix list is space-separated. Namespace URIs that are used are taken from *XPathNamespaces* property.

Constructors

C# VB.NET

```
public XMLReference();
```

Creates a new XML reference element.

C# VB.NET

```
public XMLReference(string ID);
```

Creates a new XML reference element from its *ID* .

C# VB.NET

```
public XMLReference(string ID, string URI);
```

Creates a new XML reference element from its *ID* and *URI* reference to the external data.

Config Settings (XAdESSigner Component)

The component accepts one or more of the following *configuration settings*. Configuration settings are similar in functionality to properties, but they are rarely used. In order to avoid "polluting" the property namespace of the component, access to these *internal properties* is provided through the Config method.

XAdESSigner Config Settings

AddAllDataObjectsTimestamp: Whether to add all data objects timestamp during signing.

If this property is set to True, the all data objects timestamp (xades:AllDataObjectsTimeStamp element) will be added.

ClaimedRolesXML: The XML content of the claimed roles.

Use this property to get/specify the XML content of the claimed roles element.

ClaimedRoleText: The text of the claimed role.

Use this property to get/specify the text of the first claimed role.

CommitmentTypeIndicationAllSignedDataObjects[Index]: Specifies the CommitmentTypeIndication's AllSignedDataObjects.

This property contains if the CommitmentTypeIndication's AllSignedDataObjects element is present that indicates that all the signed data objects share the same commitment. Index value could be omitted for the first CommitmentTypeIndication element.

CommitmentTypeIndicationCount: The number of the CommitmentTypeIndication elements.

Returns the number of the xades:CommitmentTypeIndication elements available.

CommitmentTypeIndicationIdentifier[Index]: Specifies the CommitmentTypeIndication's CommitmentTypeId's Identifier.

This property contains an identifier indicating the type of commitment made by the signer in the CommitmentTypeIndication's CommitmentTypeId's Identifier element. Index value could be omitted for the first CommitmentTypeIndication element.

CommitmentTypeIndicationIdentifierDescription[Index]: Specifies the CommitmentTypeIndication's CommitmentTypeId's Description.

This property contains an identifier's description indicating the type of commitment made by the signer in the CommitmentTypeIndication's CommitmentTypeId's Description element. Index value could be omitted for the first CommitmentTypeIndication element.

CommitmentTypeIndicationIdentifierDocumentationReferences[Index]: Specifies the CommitmentTypeIndication's CommitmentTypeId's DocumentationReferences.

This property contains an identifier's documentation references indicating the type of commitment made by the signer in the CommitmentTypeIndication's CommitmentTypeId's DocumentationReferences element. Index value could be omitted for the first CommitmentTypeIndication element.

CommitmentTypeIndicationIdentifierQualifier[Index]: Specifies the CommitmentTypeIndication's CommitmentTypeId's IdentifierQualifier.

This property contains an identifier qualifier indicating the type of commitment made by the signer in the CommitmentTypeIndication's CommitmentTypeId's IdentifierQualifier element. Index value could be omitted for the first CommitmentTypeIndication element.

CommitmentTypeIndicationObjectReference[Index]: Specifies the CommitmentTypeIndication's ObjectReference.

This property contains the CommitmentTypeIndication's ObjectReference elements that refer to one or several ds:Reference elements of the ds:SignedInfo corresponding with one data object qualified by this property. Index value could be omitted for the first CommitmentTypeIndication element.

CommitmentTypeIndicationQualifiersXML[Index]: The XML content of the CommitmentTypeIndication's Qualifiers.

This property contains the CommitmentTypeIndication's Qualifiers elements XML content. Index value could be omitted for the first CommitmentTypeIndication element.

DataObjectFormatCount: The number of the DataObjectFormat elements.

Returns the number of the xades:DataObjectFormat elements available.

DataObjectFormatDescription[Index]: Specifies the DataObjectFormat's Description.

This property contains textual information related to the signed data object in the DataObjectFormat's Description element. Index value could be omitted for the first DataObjectFormat element.
DataObjectFormatEncoding[Index]: Specifies the DataObjectFormat's Encoding.

This property contains an indication of the encoding format of the signed data object in the DataObjectFormat's Encoding element. Index value could be omitted for the first DataObjectFormat element.

DataObjectFormatMimeType[Index]: Specifies the DataObjectFormat's MimeType.

This property contains an indication of the MIME type of the signed data object in the DataObjectFormat's MimeType element. Index value could be omitted for the first DataObjectFormat element.

DataObjectFormatObjectIdentifier[Index]: Specifies the DataObjectFormat's ObjectIdentifier's Identifier.

This property contains an identifier indicating the type of the signed data object in the DataObjectFormat's ObjectIdentifier's Identifier element. Index value could be omitted for the first DataObjectFormat element.

DataObjectFormatObjectIdentifierDescription[Index]: Specifies the DataObjectFormat's ObjectIdentifier's Description.

This property contains an identifier's description indicating the type of the signed data object in the DataObjectFormat's ObjectIdentifier's Description element. Index value could be omitted for the first DataObjectFormat element.

DataObjectFormatObjectIdentifierDocumentationReferences[Index]: Specifies the DataObjectFormat's ObjectIdentifier's DocumentationReferences.

This property contains an identifier's documentation references indicating the type of the signed data object in the DataObjectFormat's ObjectIdentifier's DocumentationReferences element. Index value could be omitted for the first DataObjectFormat element.

DataObjectFormatObjectIdentifierQualifier[Index]: Specifies the DataObjectFormat's ObjectIdentifier's IdentifierQualifier.

This property contains an identifier qualifier indicating the type of the signed data object in the DataObjectFormat's ObjectIdentifier's IdentifierQualifier element. Index value could be omitted for the first DataObjectFormat element.

DataObjectFormatObjectReference[Index]: Specifies the DataObjectFormat's ObjectReference.

This property contains the DataObjectFormat's ObjectReference element that reference the ds:Reference element of the ds:Signature corresponding with the data object qualified by this property.

For example, if the corresponding ds:Reference element has an Id "reference-id-1", then you should set this property to "#reference-id-1" value.

Index value could be omitted for the first DataObjectFormat element.
DataType: Specifies the external data type.

Use this property to specify the type of the external data (either DataFile, DataStream or DataBytes properties) for component.

The following data types are supported:

"" or "XML"	an XML document (by default).
"data"	a binary data.

DetachedResourceURI: Specifies a detached resource URI.

Specifies a URI used for data being signed, usually the data filename if stored along with a detached signature.

EnvelopingObjectEncoding: Specifies the enveloping object encoding.

In case of enveloping signature, this property contains the Encoding attribute of the enveloped object.

EnvelopingObjectID: Specifies the enveloping object identifier.

In case of enveloping signature, this property contains the identifier (ID) attribute of the enveloped object.

EnvelopingObjectMimeType: Specifies the enveloping object MIME type.

In case of enveloping signature, this property contains the MIME type attribute of the enveloped object.

ExclusiveCanonicalizationPrefix: Specifies the exclusive canonicalization prefix.

Specifies the prefix for the ec:InclusiveNamespaces element for the exclusive canonicalization.

Default value is "ec". In this case "ec:" prefix will be used.

Special values:

"#default" or ""	indicates that the prefix will be omitted.
"#auto"	indicates that the prefix will be auto-detected based on the parent nodes.

HMACKey: The key value for HMAC.

Use this property to set the HMAC key. The component uses base16 (hex) encoding for this configuration value.

HMACOutputLength: Sets the length of the HMAC output.

Use this property to configure the length of the HMAC output, in bytes.

HMACSigningUsed: Whether to use HMAC signing.

Set this property to true to make the component perform signing using HMAC method, rather than asymmetric cryptography. TBD

IDAttributeName: Specifies the custom name of ID attribute.

This property contains the custom name of identifier (ID) attribute. Used to identify the target XML element when reference URI has "#id_name" value or when ID attribute should be auto-generated for a target XML element.

IDAttributeNamespaceURI: Specifies the custom namespace URI of ID attribute.

This property contains the custom namespace URI of identifier (ID) attribute. Used to identify the target XML element when reference URI has "#id_name" value or when ID attribute should be auto-generated for a target XML element.

IgnoreTimestampFailure: Whether to ignore time-stamping failure during signing.

If this property is set to True, any failure during time-stamping process will be ignored.

IncludeKey: Specifies whether to include the signing key to the signature.

Set this property to True to include the public part of the signing key to the signature.

IncludeKeyValue: Specifies whether the key value must be included to the signature.

Set this property to True if the key value (its public part) should be included to the signature.

IncludeKnownRevocationInfoToSignature: Whether to include custom revocation info to the signature.

This property specifies whether revocation pieces provided via KnownCertificates, KnownCRLs, and KnownOCSPs properties should be included into the signature. This property lets you include custom validation elements to the signature in addition to the ones comprising the signing chain.

InclusiveNamespacesPrefixList: Specifies the InclusiveNamespaces PrefixList.

Use this property to read/specify InclusiveNamespaces PrefixList for exclusive canonicalization transform of SignedInfo element. See XML-Signature Syntax and Processing specification for details.

InputType: Specifies the Input type.

Use this property to specify the type of the input (either InputFile, InputStream or InputBytes properties) for component for Sign method. The following input types are supported:

"" or "XML"	an XML document (by default).
"data"	a binary data.
"base64"	Base64 encoded binary data (input data will be encoded in Base64 and will be placed in ds:Object for Enveloping signature type)

What input types could be used depends on SignatureType:

Enveloped signature type supports only an XML document as the input.
Enveloping signature type supports all types of the input.
Detached signature type supports an XML document and a binary data as the input.

InsertBeforeXMLElement: Defines the reference XML element for signature insertion.

Use this property to specify XML element before which the signature should be inserted, but it must be under the element specified by the XMLElement field.

Supported values are:

""	an empty string indicates the Document element
"#id"	indicates an XML element with specified Id
XPath expression	indicates an XML element selected using XPath expression. Use AddKnownNamespace method to specify Prefixes and NamespaceURIs For example: "/root/data[1]" - indicates the second "data" element under the document element with a name "root" "//ns1:data" - indicates a data element. "ns1" prefix should be defined via AddKnownNamespace method.
Node name	indicates an XML element selected using its NodeName. For example: "data" - indicates an XML element with node name "data".

KeyInfoCustomXML: The custom XML content for KeyInfo element.

Use this property to specify the custom XML content of the ds:KeyInfo element.

The empty elements in the custom XML content act as a placeholder for auto-generated elements.

For example to change the order of ds:KeyValue and ds:X509Data auto-generated elements use the value: "<X509Data/><KeyValue/>"

KeyInfoDetails: Specifies the signing key info details to include to the signature.

Contains a comma-separated list of values that specifies which signing key info details to include to the signature.

Supported values are:

certificate	Base64-encoded [X509v3] certificate is placed to the signature
issuerserial	X.509 issuer distinguished name/serial number pair are placed to the signature
subjectname	X.509 subject distinguished name is placed to the signature
ski	Base64 encoded plain (i.e. non-DER-encoded) value of a X509 V.3 SubjectKeyIdentifier extension is placed to the signature
crl	Base64-encoded certificate revocation list (CRL) is placed to the signature

KeyInfoID: Specifies the ID for KeyInfo element.

This property contains the identifier (ID) attribute of the ds:KeyInfo element.

KeyInfoX509Chain: Specifies which certificates from SigningChain are included in the ds:KeyInfo element.

Use this property to get or set the list of indices (relative to the SigningChain property) of the certificates to include in the ds:KeyInfo/ds:X509Data element. The value may also be the keywords *none* (include none) or *all* (include the entire chain). The default is *all*.

KeyName: Contains information about the key used for signing.

The KeyName element contains a string value (with significant whitespaces) which may be used by the signer to communicate a key identifier to the recipient. Typically, the KeyName element contains an identifier related to the key pair used to sign the message, but it may contain other protocol-related information that indirectly identifies a key pair. Common uses of the KeyName include simple string names for keys, a key index, a distinguished name (DN), an email address, etc.

ManifestCount: TBD.

TBD

ManifestID[i]: TBD.

TBD

ManifestObjectIndex[i]: TBD.

TBD

ManifestXML[i]: TBD.

TBD

NormalizeNewLine: Controls whether newline combinations should be automatically normalized.

If set to true (the default value), the component will normalize any non-compliant newline characters to match those appropriate for the operating system.

ObjectCount: TBD.

TBD

ObjectEncoding[i]: TBD.

TBD

ObjectID[i]: TBD.

TBD

ObjectMimeType[i]: TBD.

TBD

ObjectSignaturePropertiesCount: TBD.

TBD

ObjectSignaturePropertiesID[i]: TBD.

TBD

ObjectSignaturePropertiesObjectIndex[i]: TBD.

TBD

ObjectSignaturePropertiesXML[i]: TBD.

TBD

ObjectSignaturePropertyCount: TBD.

TBD

ObjectSignaturePropertyID[i]: TBD.

TBD

ObjectSignaturePropertyPropertiesIndex[i]: TBD.

TBD

ObjectSignaturePropertyTarget[i]: TBD.

TBD

ObjectSignaturePropertyXML[i]: TBD.

TBD

ObjectXML[i]: TBD.

TBD

PolicyDescription: signature policy description.

This property specifies the Description of the signature policy.

PolicyExplicitText: The explicit text of the user notice.

Use this property to specify the explicit text of the user notice to be displayed when the signature is verified.

PolicyUNNumbers: The noticeNumbers part of the NoticeReference CAdES attribute.

Defines the "noticeNumbers" part of the NoticeReference signature policy qualifier for CAdES-EPES.

PolicyUNOrganization: The organization part of the NoticeReference qualifier.

Defines the "organization" part of the NoticeReference signature policy qualifier for CAdES-EPES.

ProductionPlace: Identifies the place of the signature production.

The signature production place in JSON format that was included or to be included into the signature.

Sample value: '{"addressCountry": "UK", "addressLocality": "London", "postalCode": "N1 7GU", "streetAddress": "20-22 Wenlock Road"}'

PSSUsed: Whether to use RSASSA-PSS algorithm.

Although the RSASSA-PSS algorithm provides better security than a classic RSA scheme (PKCS#1-1.5), please take into account that RSASSA-PSS is a relatively new algorithm which may not be understood by older implementations. This is an alias for *UsePSS*.

QualifyingPropertiesID: Specifies the ID for QualifyingProperties element.

This property contains the identifier (ID) attribute of the xades:QualifyingProperties element.

QualifyingPropertiesObjectID: Specifies the ID for object with QualifyingProperties element.

This property contains the identifier (ID) attribute of the ds:Object element that contains xades:QualifyingProperties element.

QualifyingPropertiesReferenceCount: The number of the QualifyingPropertiesReference elements.

Returns the number of the xades:QualifyingPropertiesReference elements available.

QualifyingPropertiesReferenceID[Index]: Specifies the QualifyingPropertiesReference's ID.

This property contains an identifier (ID) attribute of the xades:QualifyingPropertiesReference element. Index value could be omitted for the first QualifyingPropertiesReference element.

QualifyingPropertiesReferenceURI[Index]: Specifies the QualifyingPropertiesReference's URI.

This property contains an URI attribute of the xades:QualifyingPropertiesReference element. Index value could be omitted for the first QualifyingPropertiesReference element.

RefsTimestampType: Specifies references timestamp type to include to the signature.

Contains a comma-separated list of values that specifies which references timestamp type to include to the signature when signature upgraded to XAdES-X or XAdES-E-X form.

Supported values are:

SigAndRefs	SigAndRefs timestamp
RefsOnly	RefsOnly timestamp

SignatureCompliance: Specifies the signature compliance mode.

Use this property to specify whether the signature is W3C's XMLDSig, or Electronic Banking Internet Communication Standard (EBICS) compliant.

Supported values are:

""	The same as "XML-DSig".
XML-DSig	The W3C's XMLDSig-compliant signature (by default).
EBICS	Electronic Banking Internet Communication Standard (EBICS) compliant signature. On signing the version is autodetected based on the document element.

EBICS_H3	Electronic Banking Internet Communication Standard (EBICS) compliant signature. The version is H3.
EBICS_H4	Electronic Banking Internet Communication Standard (EBICS) compliant signature. The version is H4.
EBICS_H5	Electronic Banking Internet Communication Standard (EBICS) compliant signature. The version is H5.

SignatureID: Specifies the ID for Signature element.

This property contains the identifier (ID) attribute of the ds:Signature element.

SignaturePrefix: Specifies the signature prefix.

Specifies the prefix for the Signature elements.

Default value is "ds". In this case "ds:" prefix will be used.

Special values:

"#default" or ""	indicates that the prefix will be omitted.
"#auto"	indicates that the prefix will be auto-detected based on the parent nodes.

SignatureValueID: Specifies the ID for SignatureValue element.

This property contains the identifier (ID) attribute of the ds:SignatureValue element.

SignedInfoID: Specifies the ID for SignedInfo element.

This property contains the identifier (ID) attribute of the ds:SignedInfo element.

SignedPropertiesID: Specifies the ID for SignedProperties element.

This property contains the identifier (ID) attribute of the xades:SignedProperties element.

SignedPropertiesReferenceCanonicalizationMethod: Specifies the canonicalization method used in SignedProperties reference.

Use this property to specify the canonicalization method for the canonicalization transform of the ds:Reference element that points to xades:SignedProperties element. Use cxcmNone value to not to include canonicalization transform in transform chain.

cxcmNone	`0`

cxcmCanon	1
cxcmCanonComment	2
cxcmExclCanon	3
cxcmExclCanonComment	4
cxcmMinCanon	5
cxcmCanon_v1_1	6
cxcmCanonComment_v1_1	7

SignedPropertiesReferenceHashAlgorithm: Specifies the hash algorithm used in SignedProperties reference.

Use this property to specify the hash algorithm to be used for the ds:Reference element that points to xades:SignedProperties element.

Supported values:

SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`

SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`

The default value is empty string, in this case, the hash algorithm specified in HashAlgorithm property is used.

SignedPropertiesReferenceID: Specifies the ID for Reference element that points to SignedProperties element.

This property contains the identifier (ID) attribute of the ds:Reference element that points to xades:SignedProperties element.

SignedPropertiesReferenceInclusiveNamespacesPrefixList: Specifies the InclusiveNamespaces PrefixList used in SignedProperties reference.

Use this property to specify InclusiveNamespaces PrefixList for exclusive canonicalization transform of the ds:Reference element that points to xades:SignedProperties element.

SignedPropertiesReferenceIndex: Specifies the index of SignedProperties reference.

Use this property to specify the reference's index for the ds:Reference element that points to xades:SignedProperties element.

SignedSignaturePropertiesID: Specifies the ID for SignedSignatureProperties element.

This property contains the identifier (ID) attribute of the xades:SignedSignatureProperties element.

SigningCertificatesChain: Specifies which certificates from SigningChain are included in the xades:SigningCertificate element.

Use this property to get or set the list of indices (relative to the SigningChain property) of the certificates to include in the xades:SigningCertificate element. The value may also be the keywords *none* (include none) or *all* (include the entire chain). The default is *all*.

SigningCertificatesHashAlgorithm: Specifies the hash algorithm used for SigningCertificates.

Use this property to specify the hash algorithm to be used for xades:SigningCertificates element.

Supported values:

SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`

SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`

The default value is empty string, in this case, the hash algorithm specified in HashAlgorithm property is used.

SigningTimeFormat: Specifies the date time format for the XAdES signing time.

This property contains the date time format for the XAdES signing time (xades:SigningTime element under xades:SignedSignatureProperties).

Supported values:

YYYY	Year (e.g. 1997)
YYYY-MM	Year and month (e.g. 1997-07)
YYYY-MM-DD	Complete date (e.g. 1997-07-16)
YYYY-MM-DDThh:mmTZD	Complete date plus hours and minutes (e.g. 1997-07-16T19:20+01:00)
YYYY-MM-DDThh:mm:ssTZD	Complete date plus hours, minutes and seconds (e.g. 1997-07-16T20:20:30Z)
YYYY-MM-DDThh:mm:ss.sTZD	Complete date plus hours, minutes, seconds and a decimal fraction of a second (e.g. 1997-07-16T20:20:30.4Z)

YYYY-MM-DDThh:mm:ss.sssTZD	Default. Complete date plus hours, minutes, seconds and a fraction of a second (e.g. 1997-07-16T20:20:30.451Z)

SigningTimeIsUTC: Specifies whether the XAdES signing time is in UTC.

This property controls how the value provided by ClaimedSigningTime is interpreted and serialized into the XAdES signing time (xades:SigningTime element under xades:SignedSignatureProperties).

Behavior:

"true"	Default. The provided time is interpreted as UTC. If SigningTimeZoneOffset is non-zero, the time is converted to the corresponding local time (UTC plus the offset) and serialized with that offset; otherwise it is serialized as UTC.
"false"	The provided time is interpreted as local time. No clock-time adjustment is performed. The value is serialized as local time; if SigningTimeZoneOffset is non-zero, the specified offset is included.

SigningTimeZoneOffset: Specifies the time zone offset for the XAdES signing time.

This property contains the time zone offset in minutes for the XAdES signing time (xades:SigningTime element under xades:SignedSignatureProperties).

SigPolicyDescription: signature policy description.

This property specifies the Description of the signature policy (an alias for PolicyDescription).

SigPolicyExplicitText: The explicit text of the user notice.

Use this property to specify the explicit text of the user notice to be displayed when the signature is verified (an alias for PolicyExplicitText);

SigPolicyHash: The EPES policy hash.

Use this configuration setting to provide the EPES policy hash.

SigPolicyHashAlgorithm: The hash algorithm that was used to generate the EPES policy hash.

Use this setting to provide the hash algorithm that was used to generate the policy hash.

SigPolicyID: The EPES policy ID.

The EPES signature policy identifier, in dotted OID format (1.2.3.4.5).

SigPolicyNoticeNumbers: The noticeNumbers part of the NoticeReference CAdES attribute.

Defines the "noticeNumbers" part of the NoticeReference signature policy qualifier for CAdES-EPES (an alias for PolicyUNNumbers).

SigPolicyNoticeOrganization: The organization part of the NoticeReference qualifier.

Defines the "organization" part of the NoticeReference signature policy qualifier for CAdES-EPES (an alias for PolicyUNOrganization).

SigPolicyURI: The EPES policy URI.

Assign the EPES policy URI to this setting.

StripWhitespace: Controls whether excessive whitespace characters should be stripped off when loading the document.

Set this property to true to have the component eliminate excessive whitespace from the document on loading.

TimestampCanonicalizationMethod: Specifies canonicalization method used in timestamp.

Use this property to specify the canonicalization method used in timestamp.

cxcmNone	`0`
cxcmCanon	`1`
cxcmCanonComment	`2`
cxcmExclCanon	`3`
cxcmExclCanonComment	`4`
cxcmMinCanon	`5`
cxcmCanon_v1_1	`6`
cxcmCanonComment_v1_1	`7`

TimestampValidationDataDetails: Specifies timestamp validation data details to include to the signature.

Contains a comma-separated list of values that specifies which validation data values details to include to the signature when xades:TimeStampValidationData element added.

Supported values are:

certificate	Base64-encoded [X509v3] certificates
crl	Base64-encoded certificate revocation lists (CRL)
ocsp	OCSP responses

UseCustomTransformOrder: Enables custom ordering for the document transforms.

Use this property to disable automated ordering of signature transforms.

UseHMACSigning: Whether to use HMAC signing.

Set this property to true to make the component perform signing using HMAC method, rather than asymmetric cryptography.

UseMultipleX509DataNodes: Controls whether certificates are serialized in multiple ds:X509Data nodes under ds:KeyInfo.

Use this property to choose the layout of certificates (from SigningCertificate and SigningChain properties) within the ds:KeyInfo element. When set to *true*, the certificates are serialized across several *ds:X509Data* nodes (e.g., one per certificate). When set to *false*, all selected certificates are serialized inside a single *ds:X509Data* node. This setting affects only the structure of *ds:X509Data*, not which certificates are included (see *KeyInfoX509Chain*). The default is *false*.

UsePSS: Whether to use RSASSA-PSS algorithm.

Although the RSASSA-PSS algorithm provides better security than a classic RSA scheme (PKCS#1-1.5), please take into account that RSASSA-PSS is a relatively new algorithm which may not be understood by older implementations.

ValidationDataRefsDetails: Specifies validation data references details to include to the signature.

Contains a comma-separated list of values that specifies which validation data references details to include to the signature when signature upgraded to XAdES-C or XAdES-E-C form.

Supported values are:

certificate	References to X.509 certificates
crl	References to certificate revocation lists (CRL)
ocsp	References to OCSP responses

ValidationDataRefsHashAlgorithm: Specifies the hash algorithm used in validation data references.

Use this property to specify the hash algorithm used to compute hashes for validation data references.

Supported values:

SB_HASH_ALGORITHM_MD5	`MD5`
SB_HASH_ALGORITHM_SHA1	`SHA1`
SB_HASH_ALGORITHM_SHA224	`SHA224`
SB_HASH_ALGORITHM_SHA256	`SHA256`
SB_HASH_ALGORITHM_SHA384	`SHA384`
SB_HASH_ALGORITHM_SHA512	`SHA512`
SB_HASH_ALGORITHM_RIPEMD160	`RIPEMD160`
SB_HASH_ALGORITHM_GOST_R3411_1994	`GOST1994`
SB_HASH_ALGORITHM_WHIRLPOOL	`WHIRLPOOL`
SB_HASH_ALGORITHM_SHA3_256	`SHA3_256`
SB_HASH_ALGORITHM_SHA3_384	`SHA3_384`
SB_HASH_ALGORITHM_SHA3_512	`SHA3_512`

The default value is empty string, in this case, the hash algorithm specified in HashAlgorithm property is used.

ValidationDataValuesDetails: Specifies validation data values details to include to the signature.

Contains a comma-separated list of values that specifies which validation data values details to include to the signature when signature upgraded to XAdES-X-L or XAdES-E-X-L form.

Supported values are:

certificate	Base64-encoded [X509v3] certificates
crl	Base64-encoded certificate revocation lists (CRL)
ocsp	OCSP responses

WriteBOM: Specifies whether byte-order mark should be written when saving the document.

Set this property to False to disable writing byte-order mark (BOM) when saving the XML document in Unicode encoding.

XAdESPrefix: Specifies the XAdES prefix.

Specifies the prefix for the XAdES elements.

Default value is "xades". In this case "xades:" prefix will be used.

Special values:

"#default" or ""	indicates that the prefix will be omitted.
"#auto"	indicates that the prefix will be auto-detected based on the parent nodes.

XAdESv141Prefix: Specifies the XAdES v1.4.1 prefix.

Specifies the prefix for the XAdES v1.4.1 elements.

Default value is "xadesv141". In this case "xadesv141:" prefix will be used.

Special values:

"#default" or ""	indicates that the prefix will be omitted.
"#auto"	indicates that the prefix will be auto-detected based on the parent nodes.

XMLFormatting: Specifies the signature XML formatting.

Use this property to specify how the signature should be formatted.

Supported values:

"" or "none"	no formatting (by default).
"auto"	enables auto-formatting, equivalent to: "indent: 1; indent-char: tab; base64-max-length: 64; starting-level: node"

Custom values, contains a list of value pairs ("name:value") separated by comma or semicolon:

indent	specifies indentation level (default is 1)

indent-char	specifies indentation character: "space" or "tab" (default)
base64-max-length	specifies max length of base64 encoded data, such as signature value, certificate data and etc. (default is 64)
starting-level	specifies starting indentation level: non-negative integer or "node" - detected based on parent node, or "root" - detected based on number of parent nodes to a document element (default is "node").
indent-before-main	specifies if whitespace characters should be inserted before a main (ds:Signature) element: "auto" (default), "yes" or "no"

For more preciese formatting use OnFormatText and OnFormatElement events.
XMLSerializationCanonicalizationMethod: Specifies the canonicalization method to use for serialization.

Use this property to specify how the XML document should be serialized.

Supported values:

- cxcmNone (0)
- cxcmCanon (1)
- cxcmCanonComment (2)
- cxcmExclCanon (3)
- cxcmExclCanonComment (4)
- cxcmMinCanon (5)
- cxcmCanon_v1_1 (6)

XMLSerializationMode: Specifies the serialization mode for the extracted part of XML document.

Use this property to specify how the document piece should be serialized by GetInnerXML and GetOuterXML methods.

Supported values:

"completefragment"	as complete fragment (the default setting).
"rawtree"	as a raw tree

Base Config Settings

ASN1UseGlobalTagCache: Controls whether ASN.1 module should use a global object cache.

This is a performance setting. It is unlikely that you will ever need to adjust it.

AssignSystemSmartCardPins: Specifies whether CSP-level PINs should be assigned to CNG keys.

This is a low-level tweak for certain cryptographic providers. It is unlikely that you will ever need to adjust it.

CheckKeyIntegrityBeforeUse: Enables or disable private key integrity check before use.

This global property enables or disables private key material check before each signing operation. This slows down performance a bit, but prevents a selection of attacks on RSA keys where keys with unknown origins are used.

You can switch this property off to improve performance if your project only uses known, good private keys.

CookieCaching: Specifies whether a cookie cache should be used for HTTP(S) transports.

Set this property to enable or disable cookies caching for the component.

Supported values are:

off	No caching (default)
local	Local caching
global	Global caching

Cookies: Gets or sets local cookies for the component.

Use this property to get cookies from the internal cookie storage of the component and/or restore them back between application sessions.

DefDeriveKeyIterations: Specifies the default key derivation algorithm iteration count.

This global property sets the default number of iterations for all supported key derivation algorithms. Note that you can provide the required number of iterations by using properties of the relevant key generation component; this global setting is used in scenarios where specific iteration count is not or cannot be provided.

DNSLocalSuffix: The suffix to assign for TLD names.

Use this global setting to adjust the default suffix to assign to top-level domain names. The default is *.local*.

EnableClientSideSSLFFDHE: Enables or disables finite field DHE key exchange support in TLS clients.

This global property enables or disables support for finite field DHE key exchange methods in TLS clients. FF DHE is a slower algorithm if compared to EC DHE; enabling it may result in slower

connections.

This setting only applies to sessions negotiated with TLS version 1.3.
GlobalCookies: Gets or sets global cookies for all the HTTP transports.

Use this property to get cookies from the GLOBAL cookie storage or restore them back between application sessions. These cookies will be used by all the components that have its *CookieCaching* property set to "global".

HardwareCryptoUsePolicy: The hardware crypto usage policy.

This global setting controls the hardware cryptography usage policy.

Supported Values:

auto	Use hardware cryptography if available; otherwise, fall back to software-based cryptography (default).
enable	Always attempt to use hardware cryptography. If unavailable, exception will be thrown.
disable	Do not use hardware cryptography.

HttpUserAgent: Specifies the user agent name to be used by all HTTP clients.

This global setting defines the User-Agent field of the HTTP request provides information about the software that initiates the request. This value will be used by all the HTTP clients including the ones used internally in other components.

HttpVersion: The HTTP version to use in any inner HTTP client components created.

Set this property to 1.0 or 1.1 to indicate the HTTP version that any internal HTTP clients should use.

IgnoreExpiredMSCTLSigningCert: Whether to tolerate the expired Windows Update signing certificate.

It is not uncommon for Microsoft Windows Update Certificate Trust List to be signed with an expired Microsoft certificate. Setting this global property to true makes SBB ignore the expired factor and take the Trust List into account.

ListDelimiter: The delimiter character for multi-element lists.

Allows to set the delimiter for any multi-entry values returned by the component as a string object, such as file lists. For most of the components, this property is set to a newline sequence.

LogDestination: Specifies the debug log destination.

Contains a comma-separated list of values that specifies where debug log should be dumped.

Supported values are:

file	File
console	Console
systemlog	System Log (supported for Android only)
debugger	Debugger (supported for VCL for Windows and .Net)

LogDetails: Specifies the debug log details to dump.

Contains a comma-separated list of values that specifies which debug log details to dump.

Supported values are:

time	Current time
level	Level
package	Package name
module	Module name
class	Class name
method	Method name
threadid	Thread Id
contenttype	Content type
content	Content
all	All details

LogFile: Specifies the debug log filename.

Use this property to provide a path to the log file.

LogFilters: Specifies the debug log filters.

Contains a comma-separated list of value pairs ("name:value") that describe filters.

Supported filter names are:

exclude-package	Exclude a package specified in the value
exclude-module	Exclude a module specified in the value
exclude-class	Exclude a class specified in the value
exclude-method	Exclude a method specified in the value
include-package	Include a package specified in the value
include-module	Include a module specified in the value
include-class	Include a class specified in the value
include-method	Include a method specified in the value

LogFlushMode: Specifies the log flush mode.

Use this property to set the log flush mode. The following values are defined:

none	No flush (caching only)
immediate	Immediate flush (real-time logging)
maxcount	Flush cached entries upon reaching LogMaxEventCount entries in the cache.

LogLevel: Specifies the debug log level.

Use this property to provide the desired debug log level.

Supported values are:

none	None (by default)
fatal	Severe errors that cause premature termination.
error	Other runtime errors or unexpected conditions.
warning	Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong".
info	Interesting runtime events (startup/shutdown).
debug	Detailed information on flow of through the system.

trace	More detailed information.

LogMaxEventCount: Specifies the maximum number of events to cache before further action is taken.

Use this property to specify the log event number threshold. This threshold may have different effects, depending on the rotation setting and/or the flush mode.

The default value of this setting is 100.

LogRotationMode: Specifies the log rotation mode.

Use this property to set the log rotation mode. The following values are defined:

none	No rotation
deleteolder	Delete older entries from the cache upon reaching LogMaxEventCount
keepolder	Keep older entries in the cache upon reaching LogMaxEventCount (newer entries are discarded)

MaxASN1BufferLength: Specifies the maximal allowed length for ASN.1 primitive tag data.

This global property limits the maximal allowed length for ASN.1 tag data for non-content-carrying structures, such as certificates, CRLs, or timestamps. It does not affect structures that can carry content, such as CMS/CAdES messages. This is a security property aiming at preventing DoS attacks.

MaxASN1TreeDepth: Specifies the maximal depth for processed ASN.1 trees.

This global property limits the maximal depth of ASN.1 trees that the component can handle without throwing an error. This is a security property aiming at preventing DoS attacks.

OCSPHashAlgorithm: Specifies the hash algorithm to be used to identify certificates in OCSP requests.

This global setting defines the hash algorithm to use in OCSP requests during chain validation. Some OCSP responders can only use older algorithms, in which case setting this property to SHA1 may be helpful.

OldClientSideRSAFallback: Specifies whether the SSH client should use a SHA1 fallback.

Tells the SSH client to use a legacy ssh-rsa authentication even if the server indicates support for newer algorithms, such as rsa-sha-256. This is a backward-compatibility tweak.

PKICache: Specifies which PKI elements (certificates, CRLs, OCSP responses) should be cached.

The PKICache setting specifies which Public Key Infrastructure (PKI) elements should be cached to optimize performance and reduce retrieval times. It supports comma-separated values to indicate the specific types of PKI data that should be cached.

Supported Values:

certificate	Enables caching of certificates.
crl	Enables caching of Certificate Revocation Lists (CRLs).
ocsp	Enables caching of OCSP (Online Certificate Status Protocol) responses.

Example (default value):

Copy

```
PKICache=certificate,crl,ocsp
```

In this example, the component caches certificates, CRLs, and OCSP responses.
PKICachePath: Specifies the file system path where cached PKI data is stored.

The PKICachePath setting defines the file system path where cached PKI data (e.g., certificates, CRLs, OCSP responses and Trusted Lists) will be stored. This allows the system to persistently save and retrieve PKI cache data, even across application restarts.

The default value is an empty string - no cached PKI data is stored on disk.

Example:

Copy

```
PKICachePath=C:\Temp\cache
```

In this example, the cached PKI data is stored in the C:\Temp\cache directory.

ProductVersion: Returns the version of the SecureBlackbox library.

This property returns the long version string of the SecureBlackbox library being used (major.minor.build.revision).

ServerSSLDHKeyLength: Sets the size of the TLS DHE key exchange group.

Use this property to adjust the length, in bits, of the DHE prime to be used by the TLS server.

StaticDNS: Specifies whether static DNS rules should be used.

Set this property to enable or disable static DNS rules for the component. Works only if *UseOwnDNSResolver* is set to *true*.

Supported values are:

none	No static DNS rules (default)
local	Local static DNS rules
global	Global static DNS rules

StaticIPAddress[domain]: Gets or sets an IP address for the specified domain name.

Use this property to get or set an IP address for the specified domain name in the internal (of the component) or global DNS rules storage depending on the *StaticDNS* value. The type of the IP address (IPv4 or IPv6) is determined automatically. If both addresses are available, they are devided by the | (pipe) character.

StaticIPAddresses: Gets or sets all the static DNS rules.

Use this property to get static DNS rules from the current rules storage or restore them back between application sessions. If *StaticDNS* of the component is set to "*local*", the property returns/restores the rules from/to the internal storage of the component. If *StaticDNS* of the component is set to "*global*", the property returns/restores the rules from/to the GLOBAL storage. The rules list is returned and accepted in JSON format.

Tag: Allows to store any custom data.

Use this config property to store any custom data.

TLSSessionGroup: Specifies the group name of TLS sessions to be used for session resumption.

Use this property to limit the search of chached TLS sessions to the specified group. Sessions from other groups will be ignored. By default, all sessions are cached with an empty group name and available to all the components.

TLSSessionLifetime: Specifies lifetime in seconds of the cached TLS session.

Use this property to specify how much time the TLS session should be kept in the session cache. After this time, the session expires and will be automatically removed from the cache. Default value is 300 seconds (5 minutes).

TLSSessionPurgeInterval: Specifies how often the session cache should remove the expired TLS sessions.

Use this property to specify the time interval of purging the expired TLS sessions from the session cache. Default value is 60 seconds (1 minute).

UseCRLObjectCaching: Specifies whether reuse of loaded CRL objects is enabled.

This setting enables or disables the caching of CRL objects. When set to true (the default value), the system checks if a CRL object is already loaded in memory before attempting to load a new instance. If the object is found, the existing instance is reused, and its reference count is incremented to track its usage. When the reference count reaches zero, indicating that no references to the object remain, the system will free the object from memory. This setting enhances performance by minimizing unnecessary object instantiation and promotes efficient memory management, particularly in scenarios where CRL objects are frequently used.

UseInternalRandom: Switches between SecureBlackbox-own and platform PRNGs.

Allows to switch between internal/native PRNG implementation and the one provided by the platform.

UseLegacyAdESValidation: Enables legacy AdES validation mode.

Use this setting to switch the AdES component to the validation approach that was used in SBB 2020/SBB 2022 (less attention to temporal details).

UseOCSPResponseObjectCaching: Specifies whether reuse of loaded OCSP response objects is enabled.

This setting enables or disables the caching of OCSP response objects. When set to true (the default value), the system checks if a OCSP response object is already loaded in memory before attempting to load a new instance. If the object is found, the existing instance is reused, and its reference count is incremented to track its usage. When the reference count reaches zero, indicating that no references to the object remain, the system will free the object from memory. This setting enhances performance by minimizing unnecessary object instantiation and promotes efficient memory management, particularly in scenarios where OCSP response objects are frequently used.

UseOwnDNSResolver: Specifies whether the client components should use own DNS resolver.

Set this global property to false to force all the client components to use the DNS resolver provided by the target OS instead of using own one.

UseSharedSystemStorages: Specifies whether the validation engine should use a global per-process copy of the system certificate stores.

Set this global property to false to make each validation run use its own copy of system certificate stores.

UseSystemNativeSizeCalculation: An internal CryptoAPI access tweak.

This is an internal setting. Please do not use it unless instructed by the support team.

UseSystemOAEPAndPSS: Enforces or disables the use of system-driven RSA OAEP and PSS computations.

This global setting defines who is responsible for performing RSA-OAEP and RSA-PSS computations where the private key is stored in a Windows system store and is exportable. If set to true, SBB will delegate the computations to Windows via a CryptoAPI call. Otherwise, it will export the key material and perform the computations using its own OAEP/PSS implementation.

This setting only applies to certificates originating from a Windows system store.

UseSystemRandom: Enables or disables the use of the OS PRNG.

Use this global property to enable or disable the use of operating system-driven pseudorandom number generation.

XMLRDNDescriptorName[OID]: Defines an OID mapping to descriptor names for the certificate's IssuerRDN or SubjectRDN.

This property defines custom mappings between Object Identifiers (OIDs) and descriptor names. This mapping specifies how the certificate's issuer and subject information (ds:IssuerRDN and ds:SubjectRDN elements respectively) are represented in XML signatures.

The property accepts comma-separated values where the first descriptor name is used when the OID is mapped, and subsequent values act as aliases for parsing.

Syntax:

Copy

```
Config("XMLRDNDescriptorName[OID]=PrimaryName,Alias1,Alias2");
```

Where:

OID: The Object Identifier from the certificate's IssuerRDN or SubjectRDN that you want to map.

PrimaryName: The main descriptor name used in the XML signature when the OID is encountered.

Alias1, Alias2, ...: Optional alternative names recognized during parsing.

Usage Examples:

Map OID 2.5.4.5 to SERIALNUMBER:

Copy

```
Config("XMLRDNDescriptorName[2.5.4.5]=SERIALNUMBER");
```

Map OID 1.2.840.113549.1.9.1 to E, with aliases EMAIL and EMAILADDRESS:

Copy

```
Config("XMLRDNDescriptorName[1.2.840.113549.1.9.1]=E,EMAIL,EMAILADDRESS");
```

XMLRDNDescriptorPriority[OID]: Specifies the priority of descriptor names associated with a specific OID.

This property specifies the priority of descriptor names associated with a specific OID that allows to reorder descriptors in the ds:IssuerRDN and ds:SubjectRDN elements during signing.

XMLRDNDescriptorReverseOrder: Specifies whether to reverse the order of descriptors in RDN.

Specifies whether to reverse the order of descriptors in the ds:IssuerRDN and ds:SubjectRDN elements during XML signing. By default, this property is set to true (as specified in RFC 2253, 2.1).

XMLRDNDescriptorSeparator: Specifies the separator used between descriptors in RDN.

Specifies the separator used between descriptors in the ds:IssuerRDN and ds:SubjectRDN elements during XML signing. By default, this property is set to ", " value.

Trappable Errors (XAdESSigner Component)

XAdESSigner Errors

1048577	Invalid parameter (`SB_ERROR_INVALID_PARAMETER`)
1048578	Invalid configuration (`SB_ERROR_INVALID_SETUP`)
1048579	Invalid state (`SB_ERROR_INVALID_STATE`)
1048580	Invalid value (`SB_ERROR_INVALID_VALUE`)
1048581	Private key not found (`SB_ERROR_NO_PRIVATE_KEY`)
1048582	Cancelled by the user (`SB_ERROR_CANCELLED_BY_USER`)
1048583	The file was not found (`SB_ERROR_NO_SUCH_FILE`)
1048584	Unsupported feature or operation (`SB_ERROR_UNSUPPORTED_FEATURE`)

1048585	General error (`SB_ERROR_GENERAL_ERROR`)
39845889	The input file does not exist (`SB_ERROR_XML_INPUTFILE_NOT_EXISTS`)
39845890	Data file does not exist (`SB_ERROR_XML_DATAFILE_NOT_EXISTS`)
39845892	Unsupported hash algorithm (`SB_ERROR_XML_UNSUPPORTED_HASH_ALGORITHM`)
39845893	Unsupported key type (`SB_ERROR_XML_UNSUPPORTED_KEY_TYPE`)
39845895	Unsupported encryption algorithm (`SB_ERROR_XML_INVALID_ENCRYPTION_METHOD`)
39845896	XML element not found (`SB_ERROR_XML_NOT_FOUND`)
39845897	XML element has no ID (`SB_ERROR_XML_NO_ELEMENT_ID`)