

E-Signing Java Client API Implementation guide

Version: 1.7
Date: 13.11.2018

Contents

1. Introduction	3
Purpose.....	3
Intended audience	3
Referenced documentation	3
Terms and definitions	3
Acronym	3
Change log	4
2. Overview	5
E-signing client api	5
Configuration	5
3. Esignclientapi	8
Dependencies	8
Three steps	8
MerchantContext	8
ESigningFactory	10
ESigningFacade	10
Example code	14
ErrorCodes	15

1. Introduction

Purpose

This document describes the E-Signing Java Client API.

Intended audience

Intended audience is developers and architects implementing the E-Signing service.

Referenced documentation

Document	Description
TrustSignMessage XML Interface Specification	E-Signing messages describing the communication protocol for accessing E-Signing
TrustSignMessage-<version>.xsd	E-Signing XML Schema defining the supported XML message structures.
Nets TrustSign Integration Guide	This document gives an overview of the integration process, configuration data, information about eID providers and more.

Terms and definitions

Term	Description
Signing order	XML request to E-Signing Service defining documents to sign by people holding a digital ID. The term is used as an abstraction of a signing workflow where multiple persons is supposed to sign various documents.

Acronym

Acronym	Description
API	Application programming interface
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JKS	Java Key Store
RMS	E-Signing Service Request Management System
SDO	Signed Data Object
SEID	Samarbeid om Elektronisk ID
SSL	Secure Socket Layer
PKCS#12	Public Key Cryptography Standard 12 defines a keystore standard (short: P12)
XML	Extensible Markup Language

Change log

Version	Description	Date
1.0	First official version	14.03.2013
1.1	Added truststore creation description	20.06.2013
1.2	Updated trust store description	15.12.2015
1.3	Updated hostnames	31.03.2016
1.4	Updated with new request types	25.10.2016
1.5	Updated with new request type	25.11.2016

2. Overview

E-signing client api

The E-Signing service may be called with various messages. A message is an XML request and has a corresponding XML response. The request and response XML protocol is defined in the TrustSignMessage-<version>.xsd.

The E-Signing service messages vary from small XML structures to very large constructs. Implementing the complete interface may take some time. To reduce our customers' implementation time we offer the esignclientapi. The esignclientapi-<version>.jar implements the complete TrustSignMessage XML interface. The library offers the following functionality:

1. The library exposes Java objects that represent each message in the TrustSignMessage interface (InsertOrder, CancelOrder etc).
2. The library offers a Façade class which again has different methods (one for each type of message) for each TrustSignMessage type.
3. Validation of the request according to the latest TrustSignMessage XML schema
4. Signing of the requests creating enveloping XML Digital Signatures. All requests to the E-Signing service MUST be signed.
5. Validation of the response according to the TrustSignMessage XML schema
6. Error handling and interpretation of errorcodes from both RMS and TrustSignMessage.
7. Multi merchant configuration. The client system may initialize the library setting up many Merchant Contexts. Each Merchant Context must be unique and mapped to a unique merchant name and should have a unique MerchantID (provided by Nets). Please read the "*TrustSignMessage XML Interface Specification*" and the "*TrustSign integration guide*" for more information.
8. Communication between the client application and the remote E-Signing service. Configuration of client SSL keystore, SSL trust store and proxy settings if also supported. See the upcoming chapters for more details and example code.

Configuration

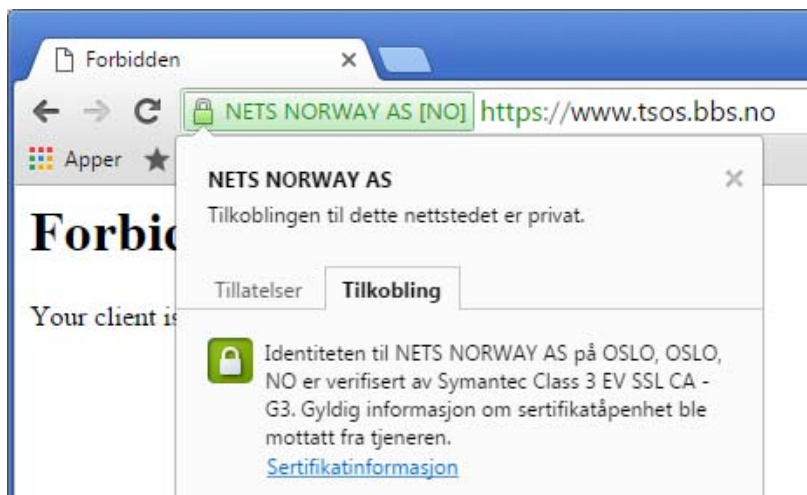
A E-Signing customer, also referred to as "merchant", is provided with the following configuration items for secure communication with the E-Signing service:

- A keystore, PKCS#12, for signing XML requests
- A keystore. PKCS#12, for secure 2-way-SSL communication (client SSL keystore)
- A MerchantID, a customer number provided by Nets. The digital certificates located in the keystores along with the MerchantID uniquely identify the customer in the E-Signing service system.

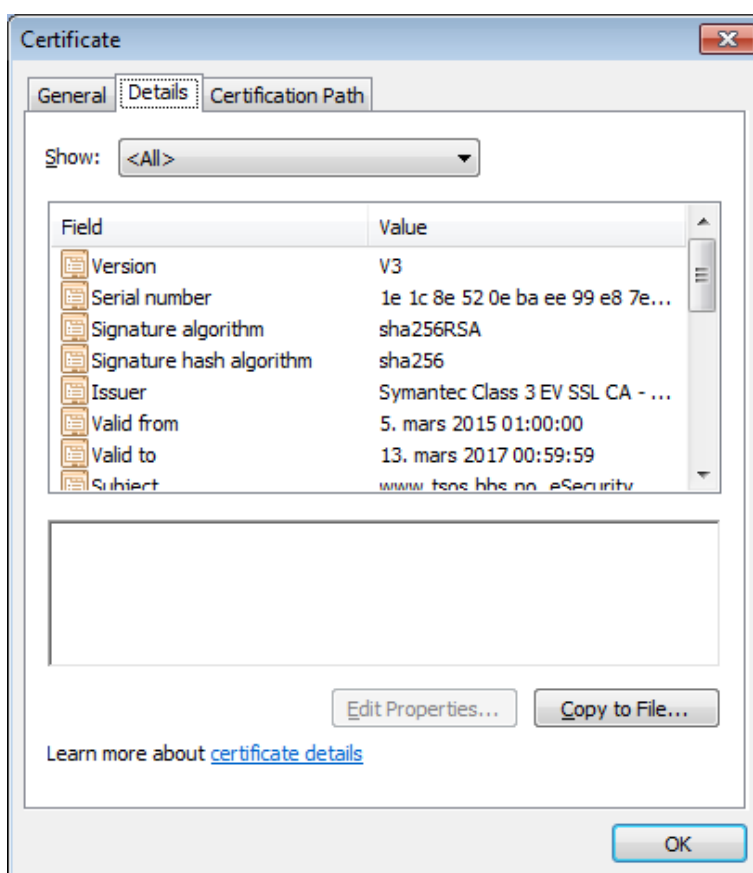
Truststore

In addition you need a truststore that contains the SSL-certificate for establishing an SSL-connection to the E-Signing server. You can create this

file by downloading the certificate or the issuer to a local certificate-file by entering the address <https://www.sign.nets.eu/> in a Chrome browser and clicking the locker-sign in the address field:



You may then open the issuer certificate using the “Certificate information” button:



Then, click on the Details-tab and select “Copy to File”. Follow the wizard instructions in order to save the certificate to a Base-64 encoded file.

Then, from the command-line, execute the following keytool-command:

```
C:> keytool -import -alias nets.es -file <cert-file> -keystore  
<your-truststore>
```

Where the alias is your description of the certificate, cert-file is the file you just stored the certificate in and your-truststore is a filename for the truststore you want to create. Answer "yes" to the question if you want to trust the certificate and choose a password for the truststore-file.

Now you have a truststore and a password that can be entered in the program-code to establish communications to the E-signing server.

Note that if you are using this document only as a form of a reference, without using the API, you may also select to trust the certificate directly as an X.509 certificate file. There are many other ways of doing this, like trusting the issuer or root certificate instead of the site certificate. The main goal is to ensure that you connect to a trusted site.

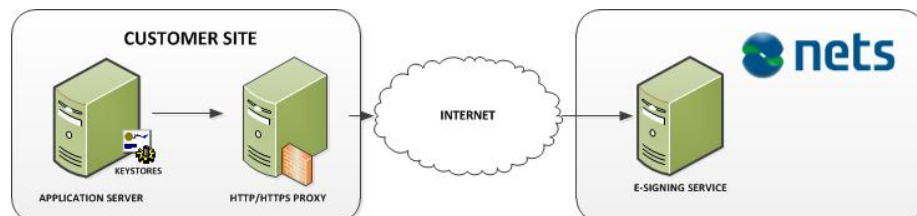
3. Esignclientapi

Dependencies

The esigningclientapi is a jar with dependencies. The following list outlines these dependencies:

- esigningclientapi-<version>.jar
- tsmxmlapi-<version>.jar
- commons-httpclient-3.1.jar
- commons-logging-1.0.4.jar
- commons-codec-1.2.jar
- bcmail-jdk16-1.40.jar
- bcprov-jdk16-1.40.jar

Alternatively you may use the esignclientapi-<version>-jar-with-dependencies.jar which is a bundle holding all the needed binaries.



Three steps

To start using the esigningclientapi include the esigningclientapi-<version>.jar along with its jar-dependencies into your project. Follow these three simple steps:

1. Construct a MerchantContext instance
2. Register the MerchantContext in the ESigningFactory
3. Obtain a ESigningFacade from the ESigningFactory. The ESigningFacade is the abstraction of the ESigningService interface.

MerchantContext

The esigningclientapi may serve multiple merchants with different credentials. Each merchant must have its own signing keys and MerchantID, each merchant must have its own context.

The MerchantContext object holds one merchant configuration. The customer application must setup a MerchantContext by calling the following setter methods on the MerchantContext object:

- setSslKeystorePath: Fully qualified filesystem path to SSL PKCS#12 keystore
- setSslKeystorePwd: The password to the SSL keystore
- setSigningKeystorePath: Fully qualified filesystem path to signing PKCS#12 keystore

- `setSigningKeystorePwd`: The password to the signing keystore
- `setTruststorePath`: Fully qualified filesystem path to SSL truststore. This is also a keystore and must hold the trusted SSL certificates. This keystore might be a JKS (Java KeyStore) or a PKCS#12 and must hold the issuer certificate of the E-Signing service Server SSL certificate.
- `setTruststorePwd`: The password to the SSL truststore
- `setTruststoreType`: Must be either `KeyStoreType.JKS` or `KeyStoreType.PKCS12`.
- `setCommTimeout`: Sets the socket timeout in milliseconds
- `setEnv`: Sets the target E-Signing service environment. Should be either `Environment.PRE_PRODUCTION` or `Environment.PRODUCTION`. The URLs to the different Environments are built into the API.
- `setHttpProxyHost`: If you have a http proxy, as shown in the figure above, then the host IP or DNS name must be set
- `setHttpProxyPort`: If a proxy host is set then setting the proxy port is mandatory
- `setHttpProxyUsername`: If HTTP proxy exists and requires basic authentication
- `setHttpProxyPassword`: If `setHttpProxyUsername` is called then the password must also be provided.
- `setMerchantId`: This is the unique merchant/customer number provided by Nets
- `setMerchantName`: The merchant name must be set and the value can be any String. The merchant name is there for you to label the merchant context with a human readable tag.

```
import no.bbs.trust.esignclientapi.impl.MerchantContext
...
MerchantContext merchantContext = new MerchantContext();

merchantContext.setSslKeystorePath("/cfg/sslkeystore.p12");
merchantContext.setSslKeystorePwd("pwd");
merchantContext.setSigningKeystorePath("/cfg/sign.p12");
merchantContext.setSigningKeystorePwd("pwd");
merchantContext.setTruststorePath("/cfg/truststore");
merchantContext.setTruststorePwd("changeit");
merchantContext.setTruststoreType(KeyStoreType.JKS);
merchantContext.setCommTimeout(5000);
merchantContext.setEnv(Environment.PRE_PRODUCTION);
merchantContext.setHttpProxyHost("proxy");
merchantContext.setHttpProxyPort("8080");
merchantContext.setHttpProxyUsername("proxyuser");
merchantContext.setHttpProxyPassword("pwd");
merchantContext.setMerchantId("1001");
merchantContext.setMerchantName("acme");
```

ESigningFactory

ESigningFactory is a MerchantContext registry. ESigningFactory exposes two methods:

1. `registerMerchantContext(MerchantContext ctx)`
This method validates the input MerchantContext to ensure that the provided merchant configuration is correct. The input MerchantContext is registered under the merchant name. **If you register two MerchantContexts with the same MerchantName then the last one registered will overwrite the existing one.**
2. `getESigningFacade(String merchantName)`
This method returns a ESigningFacade instance loaded with the MerchantContext holding the input merchantName.

```
import no.bbs.trust.esignclientapi.impl.ESigningFactory
import no.bbs.trust.esignclientapi.impl.ESigningFacade
...
ESigningFactory fac = ESigningFactory.INSTANCE;
fac.registerMerchantContext(merchantContext);

ESigningFacade facade = fac.getESigningFacade(merchantname);
```

ESigningFacade

The ESigningFacade class is obtained from the ESigningFactory and is the abstraction of the E-Signing service interface. ESigningFacade exposes the following methods:

- `InsertOrderResponse insertOrder(InsertOrderRequest req)`

Method to insert a Signing order request. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an ESignClientException is thrown holding the error code and error texts.

- `CancelOrderResponse cancelOrder(CancelOrderRequest req)`

Method to cancel an already existing order. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an ESignClientException is thrown holding the error code and error texts.

- `GetDocumentsResponse getDocuments(GetDocumentsRequest r)`

Method to retrieve Signing order documents and the status of each document. This method validates the input request object, signs the XML representation of the object, communicates with the E-

Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetNotificationLogResponse getNotificationLog(GetNotificationLogRequest request)`

Method to retrieve an existing signingorder's notification log. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetOrderResponse getOrder(GetOrderRequest request)`

Method to retrieve an existing order as it was registered. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetOrderDetailsResponse getOrderDetails(GetOrderDetailsRequest request)`

Method to retrieve various details about an existing order. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetOrdersResponse getOrders(GetOrdersRequest request)`

Method to obtain OrderIDs based on various filters. OrderIDs and statuses are returned. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetOrderStatusResponse getOrderStatus(GetOrderStatusRequest request)`

Method to retrieve all statuses of an order. OrderStatus, DocumentStatuses, SignerStatuses etc. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetSDOResponse getSDO(GetSDORequest request)`

Method to fetch the SDO (Signed Data Object) for a given OrderId. If the order is not Complete then the SDO is not returned. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetSDODetailsResponse getSDODetails(GetSDODetailsRequest request)`

Method to validate and parse a SDO. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetSigningProcessesResponse getSigningProcesses(GetSigningProcessesRequest request)`

Method to get details about a given order's signing processes. Statuses and signing URLs are provided. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `MergeSDOsResponse mergeSDOs(MergeSDOsRequest request)`

Method to merge two SDOs. Both SDOs MUST hold the same document. This is a service to merge two SDO constructs where each SDO has different signatures but over the same document. The signatures are merged into one SDO and the SDO is re-sealed leaving it untouchable. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `ModifyOrderDeadlineResponse modifyOrderDeadline(ModifyOrderDeadlineRequest request)`

Method to shift ALL deadlines in a signingorder with the provided `ShiftValue` in hours. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `ModifySigningProcessResponse modifySigningProcess(ModifySigningProcessRequest request)`

Method to set a signingorder status to "RejectedBySigner". This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `ValidateSDOResponse validateSDO(ValidateSDORequest req)`

Method to validate a SDO. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetPAdESResponse getPAdES(GetPAdESRequest req)`

Method to get the PAdES signed PDF document from input orderID after an order is completed. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GeneratePAdESResponse generatePAdES(GeneratePAdESRequest req)`

Method to generate PAdES signed PDF document from the input SDO object. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `DeleteDocumentDataResponse deleteDocumentData>DeleteDocumentDataRequest req)`

Method to delete document data and SDO from the E-Signing database. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `ModifySignerResponse modifySigner(ModifySignerRequest req)`

Method to alter the signer data such as name and signerid on an existing order that signing has not yet been started upon. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing Service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `FinalizeOrderResponse finalizeOrder(FinalizeOrderRequest req)`

Method to set an order with at least one signature to complete status. This method validates the input request object, signs the XML representation of the object, communicates with the E-Signing service and returns a response object. If an error occurs then an `ESignClientException` is thrown holding the error code and error texts.

- `GetSignatureResponse getSignature(GetSignatureRequest req)`

Method to retrieve signature for a signing along with OCSP and SSN if it is available. This method validates the input request object, signs the XML representation of the object, communicates

with the E-Signing service and returns a response object. If an error occurs then an ESignClientException is thrown holding the error code and error texts.

Example code

Below is an example of how to insert a Signing order. Creating a registering MerchantContexts was shown in the previous chapters.

```
ESigningFacade facade = ESigningFactory.INSTANCE.getESigningFacade(merch);

InsertOrderRequest request = new InsertOrderRequest();
request.setMerchantID(1001);
request.setMessageID("esignclientapi-test-1");
request.setTime(new Date());
request.setOrderID("1001-12345");
request.setAdditionalInfo("Additional info"); //Optional

String doctosign = new String("Purchased for 500 NOK. Item #34.");
Text txt = new Text();
txt.setB64DocumentBytesAsString(new
String(Base64.encode(doctosign.getBytes("UTF-8"))));

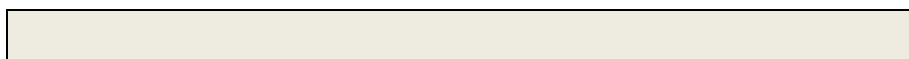
Doc doc = new Doc();
doc.setDocType(txt);
doc.setDesc("ACME Purchase order 5434");
doc.setLocalDocRef("#34");
doc.setTitle("ACME purchase order");
doc.setRequiresAuthentication(false);

// Add Docs to sign
ArrayList<Doc> documents = new ArrayList<Doc>();
documents.add(doc);
request.setDocuments(documents);

// Add Signers with preset accepted PKIs (BankID)
BankID bankid = new BankID();
SignerID signerId = new SignerID();
signerId.setIdType("SSN");
signerId.setIdValue("01017000000");
bankid.setSignerID(signerId);
ArrayList<IPKI> acceptedPKIs = new ArrayList<IPKI>();
acceptedPKIs.add(bankid);
Signer signer = new Signer();
signer.setLocalSignerRef("01017000000");
signer.setName("Ola Nordmann");
signer.setAcceptedPKIs(acceptedPKIs);
ArrayList<Signer> signers = new ArrayList<Signer>();
signers.add(signer);
request.setSigners(signers);

// ExecutionDetails
ExecutionDetails executionDetails = new ExecutionDetails();
executionDetails.setOrderDeadline(new Date(System.currentTimeMillis() +
86400000));
executionDetails.setDisplayProcessInfo("NameStatusTime");
Step step1 = new Step();
step1.setStepNumber(1);
SigningProcess signingProcess = new SigningProcess();
signingProcess.setLocalDocumentReferance("#34");
signingProcess.setLocalSignerReferance("01017000000");
step1.addSigningProcess(signingProcess);
executionDetails.addStep(step1);
request.setExecutionDetails(executionDetails);

InsertOrderResponse response = facade.insertOrder(request);
```



ErrorCodes

The following table describes the errorcodes from the esignclientapi. The errorcodes from the E-Signing service is located in the TrustSignMessage Interface Specification.

ErrorCode	Description
1000	ESigningFactory failed to register MerchantContext. Reason: (%1)
1001	ESigningFactory failed to get ESigningFacade. Reason: (%1)
2000	ESigningFacade.insertOrder failed. Reason: (%1)
2001	ESigningFacade.cancelOrder failed. Reason: (%1)
2002	ESigningFacade.getDocuments failed. Reason: (%1)
2003	ESigningFacade.getNotificationLog failed. Reason: (%1)
2004	ESigningFacade.getOrder failed. Reason: (%1)
2005	ESigningFacade.getOrderDetails failed. Reason: (%1)
2006	ESigningFacade.getOrders failed. Reason: (%1)
2007	ESigningFacade.getOrderStatus failed. Reason: (%1)
2008	ESigningFacade.getSDO failed. Reason: (%1)
2009	ESigningFacade.getSDODetails failed. Reason: (%1)
2010	ESigningFacade.getSigningProcesses failed. Reason: (%1)
2011	ESigningFacade.mergeSDOs failed. Reason: (%1)
2012	ESigningFacade.modifyOrderDeadline failed. Reason: (%1)
2013	ESigningFacade.modifySigningProcess failed. Reason: (%1)
2014	ESigningFacade.validateSDO failed. Reason: (%1)
2015	ESigningFacade.getPAdES failed. Reason: (%1)
2016	ESigningFacade.generatePAdES

	failed. Reason: (%1)
2017	ESigningFacade.deleteDocumentdata failed. Reason: (%1)
2018	ESigningFacade.finalizeOrder failed. Reason: (%1)
2019	ESigningFacade.modifySigner failed. Reason: (%1)
2020	ESigningFacade.getSignature failed. Reason: (%1)
