DIGITALISERINGSSTYRELSEN

NemLog-in

NL3 Signature SP Implementation Guidelines

Contents

Changelog

Date	Version	Change description
03-07-2020	0.1	Draft
04-08-2020	0.2	Review
1-9-2020	0.3	Minor updates
1-10-2020	0.4	Minor updates
12-11-2020	1.0	Version updated for release
3-12-2020	1.0.1	Updated based on review comments from Digst
04-01-2021	1.0.2	Support for multiple IdPs
04-02-2021	1.0.3	Clarified requirements for JWS signing keypair Updated based on review comments from Digst
09-02-2021	1.0.4	Added nemlogin-broker-mock to SignSDK, updated screenshots
29-03-2021	1.0.5	Updated based on Digst review
28-04-2021	1.0.6	Added FOCES certificate as option similar to VOCES certificates
19-05-2021	1.0.7	Minor updates Language corrections
16-11-2021	1.0.8	Updated minimum recommend iframe height. Updated screenshots
16-12-2021	1.0.9	Added [Signature Validation] as reference
15-11-2022	1.0.9	Updated layout
13-12-2022	1.0.10	Full screen signing client
18-07-2023	1.0.11	Expanded SP implementation guidelines regarding implementation of iFrame
25-09-2023	1.0.12	Support for Greenlandic language
	1.0.13	NemID references removed.
21-06-2024	1.0.14	.Net: Existing PDF manipulation framework has been replaced with PdfSharp in section 4.2 The project naming conventions in the example code have been changed. The .net version is updated to 8.0. Conversion from .html to .pdf is now handled via the Puppeteer framework. Java: Framework versions updated. No changes to documentation

References

Web Messaging	https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
eIDAS	Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. Available here: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0910&from=EN
PAdES	ETSI EN 319 142-1: AdES digital signatures; Part 1: Building blocks and PAdES baseline signatures, ETSI ESI. Available here: https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.01.01_60/en_31914201v010101p.pdf
XAdES	ETSI EN 319 132-1: XAdES digital signatures; Part 1: Building blocks and XAdES baseline signature, ETSI ESI. Available here: https://www.etsi.org/deliver/etsi_en/319100_319199/31913201/01.01.01_60/en_31913201v010101p.pdf
Format	AdES Signature Profile, https://www.ca1.gov.dk/efterlevelseserklaeringer/
Profile	Certificate Profiles, https://www.ca1.gov.dk/efterlevelseserklaeringer/

1 The Purpose and Target Audience of the Document

This document is part of the NemLog-in Service Provider Package. The purpose of this document is to serve as the technical documentation for integrating the NemLog-in signing client.

The document is aimed at developers and architects. As such it is quite technical, and the reader should be familiar with the content in the references to fully appreciate this technical document.

1.1 Document Conventions

Code examples and XML snippets are written using a fixed width font. References are marked in square-brackets, e.g. [Web Messaging].

1.2 Abbreviations

Abbreviation	Description
SP	Service Provider
SD	Signer's document. The original document that is input to the signing process. SD is in a valid Document Format.
SD Document Format	Format of Signer's Document (HTML, XML, Text or PDF)
Signature Format	The target format (XAdES, PAdES) of the signed document.
DTBS	Data To Be Signed. Intermediate format produced from the Signer's Document by SignSDK in a valid Signature Format. The DTBS is pre-signed by the SignSDK and will be signed by the NemLog-In Signing Client.
Signer	End user identity performing the actual document signing. This can be a person or an employee signing on behalf of an organization.
Signature	Signatures is produced for an individual. F.ex. a person or an employee.
Seal	Seals is a signature produced for a legal identity, f.ex. a business or organization.
JWS	JSON Web Signature https://tools.ietf.org/html/rfc7515
WYSIWYS	What You See Is What You Sign
GUUID	Global Universally Unique Identifier. For a person this is the CPR UUID and for employees this is the EmployeeUUID persistent identifier.
Signer's GUUID	Signer identifier which the SP has received in an Assertion from NemLog-in Login component as Subject->NameID attribute.
XAdES-B-LTA	XML variant of the signature format supported by the signing compont as specified in [Format].
PAdES-B-LTA	PDF variant of the signature format supported by the signing compont as specified in [Format].
Signing certificate	The qualified personal (QPerson), employee (QEmployee) or organization (QOrg) certificate with a format as specified in [Profile].

2 Introduction

This document serves as the technical documentation for how to integrate with the signature solution. SP must interact with two components, described in this document, SignSDK and signature client, for the integration.

The SPs backend uses the SignSDK which aims to validate the document to be signed for conformance and prepare a payload, containing the document and other signature parameter to be provided to the signature client in the browser.

The document to be signed is provided to the SignSDK, in this document denoted SD Document Format – are either HTML, Text, XML and PDF. The supported subsets of HTML and PDF are detailed in a subsequent chapter.

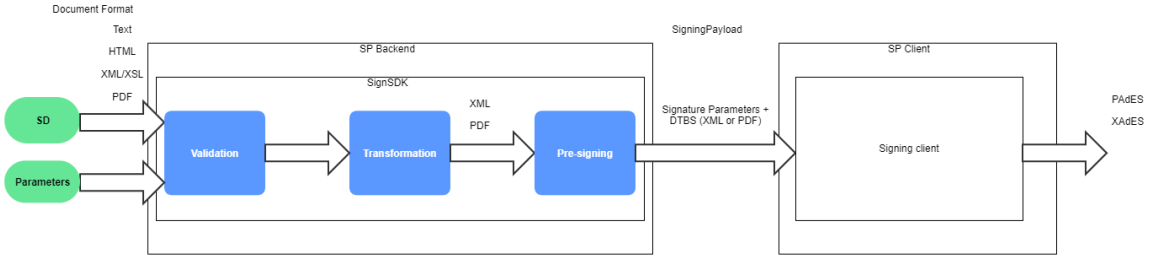
The resulting output of the signing process can be either PAdES or XAdES. All combinations are supported.

The signing process is divided into a couple of sub-processes involving the following actors:

- SP: The Service Provider wishing to sign a document.
- SP Backend: The SP backend, which will integrate with the SignSDK in order to perform first stage of the signing.
- SignSDK: The Java- or .Net-based NemLog-In SignSDK library used for first stage of the signing. The library should either be integrated directly in the SP backend, or possibly wrapped as a microservice and called by the SP backend.
- SP Web Application: The SP web application embeds the NemLog-In Signing Client in an iframe for the second stage of the signature flow.
- Signing Client. The NemLog-In Signing Client is a JavaScript Application loaded in an iframe and used in the second stage of the signature flow.
- Log-in Component. The NemLog-In Log-in component is a separate web application used by the Signing Client to authenticate the signer prior to signing the document.

First, the SP backend must use the SignSDK to transform the SD into a *Signing Payload*. The Signing Payload must then be passed on to the *NemLog-In Signing Client* running in an iframe of the SP's own web application. Lastly, the SP web application will receive the signed document from the Signing Client. The SP must also handle errors passed back from the Signing Client.

The steps are further detailed below.



Step 1: SP backend produces a Signing Payload using the SignSDK (see Chapter 4).

- As Input to the SignSDK, the SP must specify the following in a `TransformationContext` class:
  - The SD (Signer's Document) to be signed.
  - Template Signature Parameters. The Signature Parameters are detailed in a subsequent chapter and contain fields for controlling the signing process, such as document format, signature format, reference text, etc.
  - SP Signature Keys. The SP must have a VOCES or FOCES keypair (private key and certificate chain), which will be used for producing a JWS-sealed version of the Signature Parameters.
  - Transformation Properties. Certain SignSDK operations, such as transforming a HTML-based SD to PDF, can be adjusted further using transformation properties.
- The SP calls the `SigningPayloadService` service of the SignSDK with the transformation context, which in turn executes the following steps:
  - Validates the SD, ensuring that the format adheres to the specification detailed later in this document (see Chapter 3.1).
  - Transforms the SD into the DTBS (Data To Be Signed) document of the requested signature format (PAdES or XAdES).
  - If the SD is of type XML and the signature format is PAdES, then the XML + XSL will be attached to the PDF.
  - Pre-signs the DTBS document with a dummy keypair. The main purpose of this step is to prepare the DTBS for actual signing by the Signing Client, and to compute format-specific digests used for validation of the DTBS.
  - Records the DTBS digests in the Signature Parameters and seals the Signature Parametes as JWS using the SP signature keys.
- The resulting DTBS and JWS-signed signature parameters are returned as a `SigningPayload`.

Step 2: SP hands over Signing Payload to the Signing Client (see Chapter 5).

- Next step is for the SP web application to load the NemLog-In Signing Client in an iframe.
- Once the Signing Client is initialized, the SP must hand over the signing payload produced using the SignSDK. The protocol is based on HTML5 postMessage and is detailed in Chapter 5.

Step 3: NemLog-In Signing Client signs the DTBS.

- The actual signing of the DTBS is now performed by the Signing Client and involves no interaction with the SP. In short, it entails:
  - Validation of the JWS-sealed Signature Parameters and of the DTBS.
  - The user is prompted to log in via the NemLog-In log-in component.
  - The DTBS is signed according to the requested format, i.e. as PAdES or XAdES.
- In the process above, the JWS-sealed Signature Parameters will be sent to the NemLog-In Signing Backend, but at no stage will the actual document being signed leave the Signing Client running in the users' web browser.

Step 3: SP receives the result.

- The SP web application will receive the signed document from the Signing Client. This again adheres to the HTML5 postMessage-based communication protocol detailed in Chapter 5.
- Alternatively, the SP web application must be prepared to receive and handle errors or cancellation from the Signing Client.

Step 4: Signature validation.

- The SP can optionally choose to perform signature validation of the signed document, by calling the NemLog-In Signature Validation API. This is documented in [Signature Validation].

3 Signing Formats and Data Models

This chapter will describe the main formats and data models involved in the NemLog-In signing component.

3.1 Signer's Document Formats

The Signer's Document (SD) is the original document to be signed. The NemLog-In Signing Component supports four types of Signer's Documents (SDs):

- Plain Text. A UTF8 plain-text document.
- HTML. An HTML document.
- XML. An XML document plus a companion XSL document. The XSLT of the two files must yield HTML.
- PDF. A PDF document.

The SDs must adhere to a set of format-specific restrictions outlined below. The SignSDK will validate most of these restrictions.

3.1.1 Signer's Document Size Restriction

SignSDK transforms if neeed the SD into the output format. As the output format is larger than the than the original input SD the following size restrictions are to be followed:

Input SD	Signature format	Size restriction
Text	PAdES	10 MB
Text	XAdES	10 MB
HTML	PAdES	10 MB
HTML	XAdES	10 MB
XML	PAdES	10 MB
XML	XAdES	10 MB
PDF	PAdES	20 MB
PDF	XAdES	20 MB

For the XML format, the size restriction applies to the actual XML, and not the companion XSLT file.

3.1.2 HTML Restrictions

The HTML and CSS whitelists that the HTML format must adhere to are found in Appendix C

3.1.3 XML Restrictions

There are the following requirements for signing XML SD:

- The XML and companion XSL must be wellformed.
- The companion XSL is required to be version 3.0.
- Import and/or include is not allowed in the XSL.
- After transformation of XML/XSL to HTML, the resulting HTML is validated according to the HTML restrictions detailed above.

3.1.4 PDF Restrictions

For security reasons, and to ensure that a PDF document can be validated and viewed for a long time after the signature has been generated, only a subset of the PDF specification is supported, and thus, not all PDF documents may be signed. Additionally, it is recommended that PDF documents used for signing comply with the PDF/A-2 standard.

A PDF whitelist has been defined, which contains the elements from the Adobe PDF specification and elements used by Microsoft Office, that are supported by the NemLog-In Signing Component. Please find the complete whitelist in Appendix D.

Fonts used in PDF-based Signer's Documents should generally be embedded in the document. However, the PDF standard defines 14 standard fonts that need not be embedded. These fonts are also listed in Appendix D.

3.2 Signature Formats

The NemLog-In Signing Component supports two Signature Formats:

- XAdES using the LTA signature level (XAdES-B-LTA). XML-based format that embeds the original Signer's Document and adds XML-Dsig elements for the signature.
- PAdES using the LTA signature level (PAdES-B-LTA). PDF-based format that adds PDF signature dictionary-related elements to the PDF.

For more details about the supported XAdES and PAdES formats, please refer to Appendix A. The appendix also includes the XSD for the XAdES document format.

3.3 View Formats

The *View Format* is defined by a combination of the SD Document format and the Signature Format – the next section will define all valid combinations. The view format controls how the DTBS is displayed in the Signing Client.

Supported View Formats:

- Text – Used for displaying plain text-based Signer's Documents.

- HTML – Used for displaying HTML- or XML-based Signer's Documents.
- PDF – Used for displaying PDF-based Signer's Documents.

The rendering of PDF documents relies on the user's web browser for display, so it is recommended to test all supported browsers to verify that a PDF document will be shown correctly.

3.4 Valid Transformations

The table below lists the valid SD-to-DTBS transformations, and the corresponding view format used to display the document in the Signing Client.

Variant	SD Format	Signature Format	View Format	Description
A	Text	XAdES	Text	The UTF-8 plain text document is Base64-encoded and inserted in the XAdES document. As input to transformation, it must also be specified whether to use a monospace font or not.
B	Text	PAdES	PDF	The UTF-8 plain text document is transformed to PDF. As input to transformation, it must also be specified whether to use a monospace font or not.
C	HTML	XAdES	HTML	The UTF-8 HTML document is Base64-encoded and inserted in the XAdES document.
D	HTML	PAdES	PDF	The UTF-8 HTML document is transformed to PDF.
E	XML	XAdES	HTML	XML and XSL documents used as input to transformation and inserted in the XAdES document.
F	XML	PAdES	PDF	XML and XSL documents used as input to transformation. The HTML resulting from the XSLT is transformed to PDF. Both XML and XSL are added to the PDF as attached files.
G	PDF	PAdES	PDF	No transformation.
H	PDF	XAdES	PDF	The PDF document is Base64-encoded and inserted in the XAdES document.

3.4.1 XAdES Transformation Handling

For the XAdES-based transformations, the original Signer's Document is Base64-encoded and inserted in a `<SignText>` element, which constitures the part of the XML that is actually signed. Depending on the SD format, the SD is inserted as either a `<PlainText>`, `<HTMLDocument>`, `<XMLDocument>` or a `<PDFDocument>` element .

A set of *Signature Properties* may also be provided as input to the transformation. These will be included in a `<Properties>` element of the `<SignText>`, and thus, they will be signed as well. The property values may either text or Base64-encoded binary values.

The generated XAdES document will have the following template. The example is based on a plain text Signer's Document, and the XML has been added spaces for improved legibility:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SignedDocument xmlns="http://dk.gov.certifikat/nemlogin/v0.0.1#"
                xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <SignText id="id-8a7f1684-cbb3-4b15-baa3-d52af0a0594c">
        <PlainText>
            <Document>
                QmFyY2Fyb2xlCgpMZXR0…IE5hdm4uCgo=
            </Document>
            <Rendering>
                <UseMonoSpaceFont>false</UseMonoSpaceFont>
            </Rendering>
        </PlainText>
        <Properties>
            <Property>
                <Key>CaseID</Key>
                <StringValue>S123423</Value>
            </Property>
            <Property>
                <Key>SomeKey</Key>
                <BinaryValue>MII...</Value>
            </Property>
        </Properties>
    </SignText>
    <ds:Signature Id="id-9241c2c9-1cce-4e0f-b3d1-c66762b4f24d"
                  xmlns:etsi141="http://uri.etsi.org/01903/v1.4.1#"
                  xmlns:etsi132="http://uri.etsi.org/01903/v1.3.2#"
                  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
       … XML-Dsig signature
    </ds:Signature>
</SignedDocument>
```

The XML-DSig `<ds:Signature>` element is added when the XAdES is pre-signed by the SignSDK. It is subsequently updated by the Signing Client.

VARIANT A EXAMPLE

```
    <SignText id="ID_signtext">
        <PlainText>
            <Document>SSBhZ3JlZQo=</Document>
```

```
            <Rendering>
                <UseMonoSpaceFont>true</UseMonoSpaceFont>
            </Rendering>
        </PlainText>
        <Properties>
            <Property>
                <Key>CaseID</Key>
                <StringValue>S123423</Value>
            </Property>
            <Property>
                <Key>SomeKey</Key>
                <BinaryValue>MII...</Value>
            </Property>
        </Properties>
    </SignText>
```

Element `UseMonoSpaceFont` is used to signal if the document should be shown in monospace font.

Remark: only this example have the Properties element for brevity reasons.

VARIANT C EXAMPLE

```
    <SignText id="ID_signtext">
        <HTMLDocument>
            <Document>PGh0bWw+PC9odG1sPgo=</Document>
        </HTMLDocument>
    </SignText>
```

VARIANT E EXAMPLE

```
    <SignText id="ID_signtext">
        <XMLDocument>
            <Document>PHhtbD4K</Document>
            <Transformation>PHhtbD48eHNsdD48L3htbD4K</Transformation>
        </XMLDocument>
    </SignText>
```

This variant includes except for the document itself also a `Transformation` element that holds the XSLT document that is used for previewing the XML content to the Signer.

VARIANT H EXAMPLE

```
<SignText ID="ID_signtext">
    <PDFDocument>
        <Document>JVBERi0xLjMKJcTl8uXrp/Og0MTGCjmd0aC...</Document>
    </PDFDocument>
</SignText>
```

3.5 Signature Parameters

The Signature Parameters is used to control the document signing, and must be provided as input to the SignSDK, when producing a signing payload.

The Signature Parameters also contain a few fields that get updated by the SignSDK and is used for e.g. document validation in the Signing Component. These fields are not described below.

Signature parameters are mandatory when not marked as optional.

Parameter	Value	Description
flowType	”ServiceProvider”/”Broker”	Can only be “ServiceProvider” in this context.
entityID	URI	A Service Provider-specific entity ID provisioned using the NemLog-In Administration Component.
documentFormat	”TEXT”/”HTML”/”PDF”/”XML”	Signer’s Document format (TEXT, HTML, XML, PDF)
signatureFormat	”XAdES” / ”PAdES”	The type of the signed document ”XAdES-B-LTA” or ”PAdES-B-LTA”.
referenceText	Clear text Maximum allowed length is 50 characters	Allows the SP to identify the document for the signer. It will be displayed in the Signing Client and when the signer authenticates.
minAge	Integer	Minimum age requirement. When possible the Signing Component will check this against the signer’s age. Optional.
preferredLanguage	”da”/”en”/”kl”	Language used in Signing Client and login. Optional. Defaults to ‘da’.
signerSubjectNameID	Clear text	The Subject NameID of the Signer identity that must successfully authenticate in the Log-in Component. Optional. If defined, only the identified Signer can sign.
ssnPersistenceLevel	”Session”/”Global”	‘Session’ means that a unique session UUID is used as the Signing certificate’s subject

Parameter	Value	Description
		serial number. If this option is used, the Lookup Service provided by EIA shall be used retrieve information on the Signer's GUUID. The option allows for privacy of the GUUID in the signed document. 'Global' means that the Signer's GUUID is used. Optional. Default is 'Session'. Using this
anonymizeSigner	Boolean	If true, the signing certificate subject name attributes are set as "cn=Pseudonym,pseudonym=Pseudonym" and "givenName" and "surname" is omitted from the subjectDN of person- and employee-certificates. Optional. Defaults to false.
acceptedCertificatePolicies	Combination of "Person", "Employee", "Organization"	Comma-delimited list. Indicates which certificate policies can be used. When using the "Organization" policy ssnPersistenceLevel "Session" is not allowed. Optional. By default any policy is accepted.

3.6 Signing Payload

The Signing Payload consists of Signature Parameters and Data To Be Signed (DTBS).

The Signature Parameters are sealed and packaged as JWS by the SignSDK when producing a Signing Payload. For that purpose, the SP must have a VOCES or FOCES keypair. The keypair must be provisioned along with an SP-specific entity ID using the Administration Component and must be provided as input to the transformation.

The keypair can be provided as either a PKCS12 or JKS keystore. When using PKCS12, the keystore may only contain a single keypair and the corresponding certificate chain.

The sealing is done with the private key, and the certificate including the public key is attached as part of the JWS. The certificate is used by the NemLog-in backend for authentication of the Service Provider and to ensure the integrity of the DTBS is intact.

Parameter	Value	Description
signatureParameters	JWS (UTF8 string)	JWS-sealed and encoded Signature Parameters.
dtbs	Base64-encoded DTBS (UTF8 string)	Base64-encoded Data To Be Signed document.

3.7 Signing Client Error

If an error is detected in the Signing Client – typically related to input validation, this is propagated back to the Service Provider (SP) web application using the HTML5 postMessage-based protocol defined in chapter 5.

The SP web application must take appropriate action, such as displaying an error message to the Signer, and must also ensure that the Signing Client running in an iframe is removed.

The Signing Client Error is defined with the following fields:

Parameter	Value	Description
httpStatusCode	Integer	If the error has occurred during a Signing Client request to the backend Signing API, the resulting HTTP status code is included. If the error has occurred internally in the Signing Client, this field is left empty.
timestamp	ISO 8601 string	Time of error.
message	String	Error message
details	List of DetailedSigningClientError	List of detailed error codes and messages.

The `DetailedSigningClientError` is defined with the following fields:

Parameter	Value	Description
errorCode	String	Error code identifier. The full list of errors codes and the associated default error messages can be found in Appendix B.
errorMessage	String	The detailed error message. This may be more specific than the default error message associated with the error code.

Example Signing Client Error:

```
{
    httpStatusCode:500,
    timestamp:'2020-07-14T10:32:02.7859719',
    message:'Invalid Signature Parameters field dtbsSignedInfo',
    details:[{
        errorCode:'SIGN001',
        errorMessage:'Invalid Signature Parameters field dtbsSignedInfo'
    }]
}
```

4 SignSDK

The purpose of the SignSDK is to provide the Service Provider a tool to prepare the Signing Payload containing the JWS-sealed Signature Parameters and the Document To Be Signed (DTBS).

The SDK contains implementations for all the formats and data models defined in the previous chapter, and services for validating Signer's Documents and producing a Signing Payload.

Two versions of the SignSDK are provided; a Java-based and a .Net-based.

4.1 Java SignSDK

The Java-based SignSDK has the following structure:

- `doc/` Documentation, guides and build instructions.
- `library/` The Service Provider libraries.
- `examples/` Example web application.
- `test/` Unit tests.
- Along with maven project files and README files.

The SignSDK library has been organized into a set of sub-projects with the aim of reducing the number of transitive dependencies for the Service Provider to a minimum for a specific flow. Each project has a readme for specific documentation.

The table below will outline which of these sub-projects should be included depending on the Signer's Document formats and Signature Formats supported by the Service Provider.

Project name	SD Format	Signature Format	Description
nemlogin-signing-core	All	All	Contains core models and service definitions. Mandatory.
nemlogin-signing-jws	All	All	JWS-sealing of Signature Parameters. Mandatory.
nemlogin-signing-xades		XAdES	Generating and pre-signing XAdES
nemlogin-signing-pades		PAdES	Generating and pre-signing PAdES
nemlogin-signing-pdf-generator	Text, HTML, XML	PAdES	HTML-to-PDF transformation used for generating PDF from Signer's Documents of type Text, HTML and XML.
nemlogin-signing-pdf-validator	PDF		Validation of Signer's Documents of type PDF.
nemlogin-signing-html-validator	HTML, XML		Validation of Signer's Documents of type HTML and XML+XSL.
nemlogin-signing-validation			Simple client API for calling the NemLog-In Validation API and validate the signature of a singed document.

Project name	SD Format	Signature Format	Description
nemlogin-signing-spring-boot			Thin wrapper of *nemlogin-signing-core* for use in Spring Boot projects

The libraries have been implemented to use the Java `ServiceLoader` technology. The libraries that are loaded on the classpath (e.g. via dependency in the maven pom.xml file), will automatically be initialized and used whenever a matching combination of SD Document Format and Signing Format is specified. Hence, if the Service Provider e.g. only ever generates XAdES from plain text documents, they should avoid all the PDF-related project dependencies, and thus, they avoid all the transitive dependencies used for producing and parsing PDF.

Similarly, the nemlog-signing-validation dependency should only be included if the SP validates the signed document, and the nemlogin-signing-spring-boot should only be included in Spring Boot projects.

As an alternative to adding the SignSDK project dependencies to their own application, the SP may consider wrapping the `SigningPayloadService` as a microservice, and then call this via a simple REST API to produce a Signing Payload.

The SignSDK also ships with a couple of projects to help illustrate how to use the SignSDK libraries.

Project name	Description
nemlogin-signing-test	Relevant tests which serve to demonstrate how to use the SignSDK library.
nemlogin-signing-webapp	This example web application written in Spring Boot illustrates how to use the SignSDK libraries and how to interact with the Signing Client through an iframe.
nemlogin-broker-mock	Another example web application written in Spring Boot but targeted to NemLog-In Signing Brokers. It illustrates via mock code how Brokers may write their own Signing Client. Service Providers should ignore this module.

4.1.1 Generate Signing Payload

Although subject to extensive customization, the main procedure for creating a Signing Payload is quite simple and outlined in the following code:

```
// Instantiate Signer's Document
String filePath = "test.pdf";
SignersDocumentFile file = SignersDocumentFile.builder()
            .setPath(Path.of(filePath))
            .build();
SignersDocument sd = new PdfSignersDocument(file);
```

```
// Instantiate Signature Parameters
SignatureParameters signatureParameters = SignatureParameters.builder()
        .setFlowType(FlowType.ServiceProvider)
        .setEntityID("https://sp-entity-id")
        .setDocumentFormat(DocumentFormat.PDF)
        .setSignatureFormat(SignatureFormat.PAdES)
        .setReferenceText("Signing " + filePath)
        .build();

// Instantiate SP keys used for JWS-sealing the signature parameters
SignatureKeys signatureKeys = new SignatureKeysLoader()
        .setKeystoreClassPath("keystore path")
        .setKeystorePassword("keystore password")
        .setKeyPairAlias("alias")
        .setPrivateKeyPassword("key password")
        .loadSignatureKeys();

// Instantiate a transformation context
TransformationContext ctx =
        new TransformationContext(sd, signatureKeys, signatureParameters);

// Generate a Signing Payload
SigningPayloadService service = new SigningPayloadService();
SigningPayloadDTO signingPayload = service.produceSigningPayloadDTO(ctx);
```

This method for producing the Signing Payload will:

- Validate the SD (Signer's Document) according to the restrictions of section 3.1.
- Transforms the SD into a DTBS (Data To Be Signed) document of the requested signature format (PAdES or XAdES).
- If the SD is of type XML and the signature format is PAdES, then the XML + XSL will be attached to the PDF.
- Pre-sign the DTBS document with a dummy keypair. The main purpose of this step is to prepare the DTBS for actual signing by the Signing Client, and to compute format-specific digests used for validation of the DTBS.
- Record the DTBS digests in the Signature Parameters and seal the Signature Parameters in JWS encoding using the SP signature keys.

The signing payload can subsequently be transferred to the Service Provider web application and passed on to the Signing Client running in an iframe, as detailed in chapter 5.

4.1.2 Instantiating Signer's Documents

The example above demonstrated how to instantiate a PDF-based Signer's Documents. Other types of documents may be instantiated using:

```
// Instantiate Signer's Document files in different ways
SignersDocumentFile htmlFile = SignersDocumentFile.builder()
    .setUrl("some url")
    .build();
SignersDocumentFile textFile = SignersDocumentFile.builder()
    .setData("Hello world".getBytes(StandardCharsets.UTF_8))
    .setName("plaintext.txt")
    .build();
SignersDocumentFile xmlFile = SignersDocumentFile.builder()
    .setClassPath("some xml classpath")
    .setName("test.xml")
    .build();
SignersDocumentFile xslFile = SignersDocumentFile.builder()
    .setClassPath("some xsl classpath")
    .setName("test.xsl")
    .build();

// Instantiate Signer's Documents of different types
SignersDocument htmlSd = new HtmlSignersDocument(htmlFile);
boolean monospace = false;
SignersDocument textSd = new PlainTextSignersDocument(textFile, monospace);
SignersDocument xmlSd = new XmlSignersDocument(xmlFile, xslFile);
```

It is also possible to set creation time and last modified time of the SD files.

4.1.3 Adding Signed Properties to XAdES

The NemLog-In XAdES signature format allows for signed properties to be added and included in the signature. See example in chapter 3.4.1. The property values may be either String-based or binary.

These properties are specified when instantiating the Signer's Document:

```
// Instantiate Signed Properties
SignProperties signProperties = new SignProperties();
signProperties.put("CaseID", new StringValue("S76SH75657"));
```

```
SignersDocument sd = new PdfSignersDocument(file, signProperties);
```

4.1.4 Specifying Transformation Properties

Certain SignSDK services, such as the service that generates PDF from HTML, will allow for *transformation properties* to customize the behaviour.

```
Properties props = new Properties();
props.put("nemlogin.signing.pdf-generator.page-size", "a5 landscape");

// Instantiate transformation context including transformation properties
TransformationContext ctx =
  new TransformationContext(sd, signatureKeys, signatureParameters, props);
```

4.1.5 Customizing PDF generation

The following transformation properties can be used to customize how PDF files are generated from TEXT, XML and HTML Signer's Documents.

All properties have a "`nemlogin.signing.pdf-generator.`" prefix, excluded for brevity below.

Property	Default value	Description
color-profile	"default"	"default" adds a default color profile. "none" adds no profile. All other values are treated as a path to a .icc file and should have protocol prefix like "classpath:" or "file:"
fonts	"default"	"default" adds support for 14 standard PDF fonts. "embed" will embed the following list of fonts.
font[x].name		Name of the x'th font to embed.
font[x].path		Path of the x'th font to embed. Should have protocol prefix like "classpath:" or "file:"
page-size	"a4 portrait"	The CSS 2.1 @page size.
page-margin	"1cm"	The CSS 2.1 @page margin.
page-style		The page-style will be injected in the HTML as a <style> element. If defined, page-size and page-margin is ignored.

EXAMPLE

```
nemlogin.signing.pdf-generator.page-size = "a5 landscape"
```

4.2 .Net SignSDK

The .Net SignSdk class library is implemented targeting .Net 8.0.

The project can be opened with Visual Studio 2022 with the included solution file xxxx.sln.

The solution has a folder structure as below:

- src – Containing demo application for the library and projects for the library
- library – Containing PDFSharp source code that has been used.
- test – Test project with examples of how to use the library.

The solution includes the following projects:

Project name	Description
Nemlogin.QualifiedSigning.SDK.Core	Core model and general logic for the library. Entry point for using the SignSdk. Contains method to produce the signing payload passed to the signing client. Validation of input formats for all signers document formats.
Nemlogin.QualifiedSigning.SDK.Pades	Project that handles PAdES pre-signing and transformation from signer document formats to PAdES.
Nemlogin.QualifiedSigning.SDK.Xades	Project that handles XAdES pre-signing and transformation from signer document formats to XAdES.
Nemlogin.QualifiedSigning.Common	Project that contains the common classes that we need to use in different projects.
Nemlogin.QualifiedSigning.Example.WebApp	Demo application with examples of how to use the library and integrating with signing client.
Nemlogin.QualifiedSigning.SDK.Tests	Test project with relevant tests that show how to use the library.

The projects in the library are logically structured with parts separating responsibility for each of the flows and minimize dependencies.

Dependencies and logic for validation, transformation etc. is abstracted behind projects and interfaces to allow the Service Provider with a minimum of requirements to replace functionality if needed.

The library targets .Net 8.0. The Service Provider will have to use the library only in .Net 8.0 or higher.

If the Service Provider is integrating the library into .Net applications it is recommended using .Net dependency injection design pattern like below example normally setup in startup.cs of you web application.

```
services.AddTransient<ISigningPayloadService, SigningPayloadService>();
```

For working example see the Nemlogin.QualifiedSigning.Example.WebApp Demo application.

Logging in the SignSdk is implemented using standard .Net logging abstrations through the Microsoft.Extensions.Logging.Abstrations interface. In the library it is implemented and used by resolving ILogger and ILoggerFactory in a static class “LoggerCreator”. This implementation can be changed to the what the Service Providers need.

4.2.1 Generate Signing Payload

For generating signing payload with the .Net SignSdk the below code shows a simple example.

The below example is a bit modified but else taken directly from the DocumentSigningService.GenerateSigningPayload method from the Nemlogin.QualifiedSigning.Common.

```
// Instantiate Signer’s Document
string filePath = “test.pdf”;
SignersDocumentFile signersDocumentFile = new SignersDocumentFileBuilder()
                    .WithName(fileName)
                    .WithPath(filePath)
                    .Build();

SignersDocument signersDocument =
new PdfSignersDocument(signersDocumentFile);

// Create Signature Parameters
SignatureParameters signatureParameters = new SignatureParametersBuilder()
             .WithFlowType(FlowType.ServiceProvider)
             .WithEntityID(“https://sp-entity-id”)
             .WithReferenceText(“Your own referencetext here”)
             .WithSignersDocumentFormat(signersDocument.DocumentFormat)
             .WithSignatureFormat(SignatureFormat.PAdES)
             .Build();

// Load the signaturekeys used for signing the JWT
SignatureKeys signatureKeys = new SignatureKeysLoader()
                 .WithKeyStorePath(“KeystorePath”)
                 .WithKeyPairAlias(“KeyPairAlias”)
                 .WithKeyStorePassword(“KeystorePassword”)
                 .WithPrivateKeyPassword(“PrivateKeyPassword”)
                 .LoadSignatureKeys();

// Instantiate TransformationContext
TransformationContext ctx =
new TransformationContext(signersDocument, signatureKeys, signatureParamete
rs);
```

```
// Generate SigningPayload – should be changed from instantiation to DI
// of this and logger in production application
SigningPayloadService signingPayloadService =
new SigningPayloadService(new NullLogger<SigningPayloadService>());
SigningPayload signingPayload = signingPayloadService.ProduceSigningPayload
(ctx);
```

4.2.2 Instantiating Signer's Documents

The example above demonstrated how to instantiate a PDF-based Signer's Documents.

In SignersDocumentLoader.cs in the Nemlogin.QualifiedSigning.Common project there are working code that create signers document based on the type/extension of the file/path input.

See example by looking at "CreateSignersDocumentFromFile" method in SignersDocumentLoader.cs.

SignersDocumentFile would normally be instantiated with the SignersDocumentFileBuilder like below:

```
SignersDocumentFile signersDocumentFile = new SignersDocumentFileBuilder()
                    .WithName("In examples app filename is used here")
                    .WithPath("Path and filename")
                    .Build();
```

You can also specify Uri to a file instead of path:

```
SignersDocumentFile signersDocumentFile = new SignersDocumentFileBuilder()
                    .WithName("In examples app filename is used here")
                    .WithUri("some url")
                    .Build();

// Instantiate Signer's Documents of different types
SignersDocument signersDocument =
new XmlSignersDocument("Xml filename", "xslt filename");

bool useMonoSpace = true;
SignersDocument signersDocument = new PlainTextSignersDocument("filename",
useMonoSpace);
```

4.2.3 Adding Signed Properties to XAdES

The NemLog-In XAdES signature format allows for signed properties to be added and included in the signature. See example in chapter 3.4.1. The property values may be either String-based or binary.

Like the Java SignSdk these properties are specified when instantiating the Signer's Document.

Below example shows how to add a stringvalue to the collection:

```
SignProperties signProperties = new SignProperties();
signProperties.Add("ExampleID",
new SignPropertyValue("GHGEV33844", SignPropertyValue.SignPropertyValueType
.StringValue));
SignersDocument signersDocument =
new PlainTextSignersDocument(signersDocumentFile, signProperties);
```

4.2.4 Customizing PDF generation

The following transformation properties can be used to customize how PDF files are generated from TEXT, XML and HTML Signer's Documents.

All properties have a "nemlogin.signing.pdf-generator." prefix, excluded for brevity below.

Property	Default value	Description
color-profile	"default"	"default" adds a default color profile. If not specified not profile is added. All other values are treated as a path to a .icc file and should specify the full path included to the .icc file
fonts	"default"	"default" adds support for 14 standard PDF fonts. "embed" will embed the following list of fonts.
font[x].name		Name of the x'th font to embed.
font[x].path		Full path of the x'th font to embed.
page-size	"A4"	Sets the page-size for Pdfharp to use for PDF generation.
page-orientation	"portrait"	Sets the page orientation to either "portrait" or "landscape".
page-margin	"1cm"	Page margin assigned in centimeters.

4.2.5 External dependencies

Document validation and transformations are done using external libraries which are all included with the sourcecode within the solution folder.

The below is brief description of external libraries that are used in the .Net SignSdk.

PDFSHARP

From the folder path of the solution file, you will have a folder name "PdfSharp".
This folder includes sourcecode for a port of two of the original PdfSharp projects to .Net 6.0. These two projects are used in the .Net SignSdk library.
The original PdfSharp sourcecode can be found here: https://github.com/KDS/PDFsharp/tree/pdfsharp-extended

Project used in SignSdk.Net are:
PdfSharp – Core package for PdfSharp to work.
PdfSharp provides diverse ways of configuring transformations to PDF. For details refer to the documentation of PdfSharp.
If the service provider wants to replace or change implementations regarding the external packages and the use in the SignSdk then look at the direct implementations of the "ITransformator" and "IValidator" interfaces in the SignSdk library.

HTMLAGILITYPACK

HTMLAgilityPack is a HTML parser used for validating HTML documents against a whitelist.

Sourcecode are also included in the folder “html-agility-pack” in the main solution folder.

For detailed information about this package please refer to the home page of the project: https://html-agility-pack.net/

PUPPETEERSHARP

Puppeteersharp is a library to run embedded browsers in the code. Further it can be headless, so it is invisible.

The example code downloads the appropriate version of Chrome depending on the platform. Note if you run on a linux image for kubenettes you might need to install appropriate dependencies before it works.

Important security note: In the context of the signSDK PuppeteerSharp is only used for files conversion purposes. However, as PuppeteerSharp has a full Chrome browser engine installed there is a theoretical risk that this could be misused to call out of the service to external hosts. It is therefore important to make sure that this risk is addressed by firewall policies or alternative security measures in the system using the signSDK.

Note: It has been observed that fonts might need to be installed on the operating system to get the same behaviour as previous versions of the signSDK. If the Chrome engine cannot find the font it will default to the system default. Other minor visual differences compared to previous signSDK versions can occur for files converted from text formats to PDF.

For detailed information about this package please refer to the home page of the project: https://www.puppeteersharp.com/

4.3 Signature Validation

SignSDK provides a simple client library for validating the signature of a signed document. The validation service calls the public NemLog-In Signature Validation API documented in [Signature Validation].

EXAMPLE USAGE:

```
// Can also be initialized from an InputStream, a Path, etc…
SignatureValidationContext ctx = SignatureValidationContext.builder()
        .setValidationServiceUrl(“https://...”)
        .setDocumentName(“signed-document-name.pdf”)
        .setDocumentData(signedDocumentBytes)
        .build();
```

```
SignatureValidationService service = new SignatureValidationService();
ValidationReport report = service.validate(ctx);
```

4.4 Error Exceptions

If during during validation or transformation an error will occur, the SignSDK will throw an appropriate exception subclassed from the base NemLogInException holding an error-code 'SDK*' and a detailed message. The error-codes are listet in Appendix B.

5 Signing Client Integration

This section is meant for Service Providers to gain a quick overview of the development effort required to integrate with the NemLog-in JavaScript Signing Client.

Once a Signing Payload has been produced by the Service Provider (SP) backend using the SignSDK (see chapter 4), the SP web application must load the Signing Client in an iframe, and then hand over the Signing Payload to the Signing Client. The SP web application parent page must also be ready to receive the signed document from the Signing Client, or indeed handle errors or cancellation, as directed by the Signing Client. Communication between the SP web application and the Signing Client iframe is handled by the Signing Client Message Protocol described later in this chapter.

The example web application of the SignSDK also illustrates how the Signing Client may be integrated in an SP web application.

The Signing Client supports Chrome and Safari browsers on the following devices.

- Desktop
  - Windows 10
  - macOS Catalina
- Mobile/Tablet
  - iOS 15.0
  - Android 11

5.1 Accessability statement

The signing session with the Signing Client has two time-outs, which cannot be extended and therefor something the user should be made aware of.

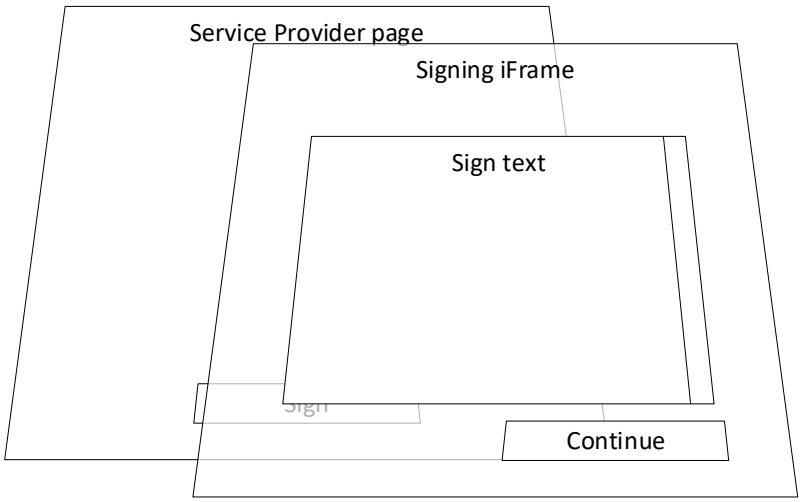
It is recommended that the Service Provider directly makes the user aware of these time-outs and also indicate them in the Service Providers accessibility statement.

	Accessability statement
1	The user has 30 minutes to read the entire document to be signed.
2	Once the document has be read, the user proceeds to NemLog-in for authentication, which must be completed within 15 minutes. I.e. the user has 15 minutes to provide authentication, credentials, accepts term and conditions, and approve the signature operation.

5.2 Iframe Integration

The Signing Client is integrated with the Service Provider's web application using an `<iframe>` element, which enables a web page to allocate a segment of its area to another page.

The content of the iframe is responsive and must *fill out the entire page* as an overlay.



NB: *Deviating from the full-screen view can affect page functionality and is not supported out of the box.*Mobile browsers have a viewport that extend beyond the size of the display, so to fill out the entire page on mobile and standard web display you must add below CSS to the IFRAME.

NB: Adding your own content around the signing window is unsupported and might not work on mobile devices.Deviating can result in double scrollbars making it difficult or impossible for the user to activate the next step in the signing flow.

Look at the reference implementation (example) in the SignSDK, and make sure that below CSS is part of your solution and let the iframe fill out the entire page.

Example CSS:

```
#signing-window {
  display: block;
  position: absolute;
  top: 0;
  left: 0;
  padding: 0;
  margin: 0;
  border: none;
  width: 100%;
  height: 100%;
}
```

Add overflow control to body to prevent scrolling outside the iframe when it is active.

```
 body {
   overflow: hidden;
  }
```

Example IFRAME:

<iframe id="signing-window" sandbox="allow-scripts allow-same-origin allow-popups allow-popups-to-escape-sandbox" src="...">
</iframe>

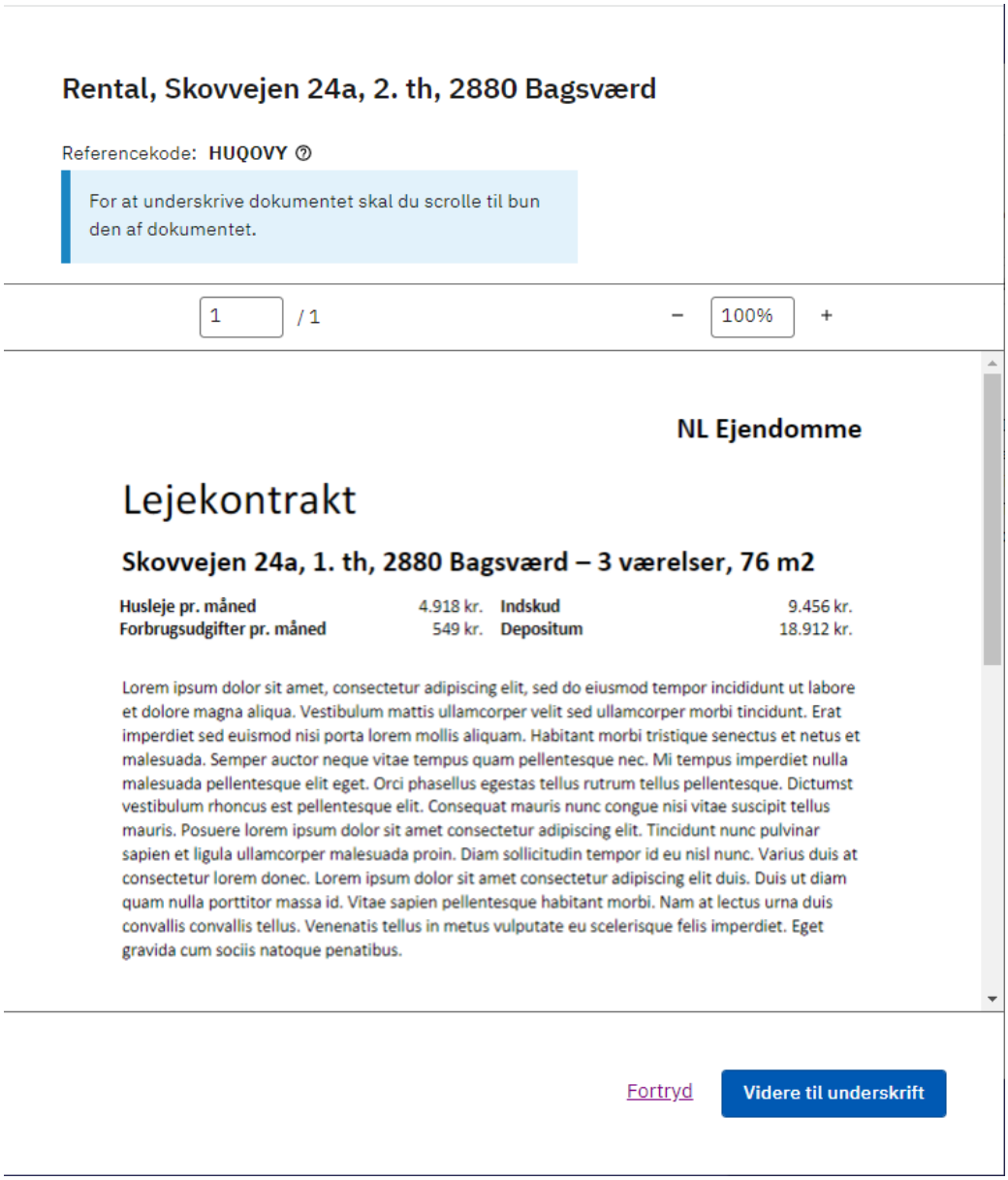


Figure 1The NemLog-in Javascript Signing Client loaded in an <iframe>. The <iframe> is everything incl. the outmost border

5.3 Signing Client Messaging Protocol

A protocol has been defined to facilitate messaging between the Signing Client running in an iframe and the SP parent page. The protocol is based on one side sending JSON objects through Window.postMessage() and the other side receiving the messages by registering a message listener. Please refer to HTML5 [Web Messaging].

The JavaScript that the SP should add to the SP page containing the Signing Client iframe, might be along the lines of:

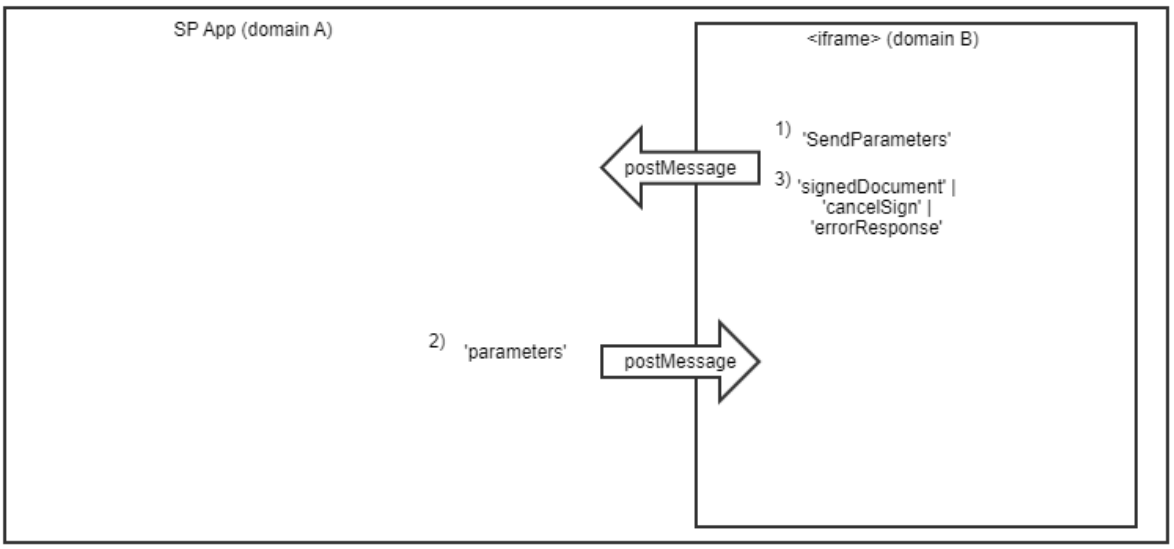
```
<script type="javascript">

  // TODO: Initialize with the Signing Payload produced by SignSDK
  var signingPayload = {};

  function onSigningClientMessage(event) {
    const win = document.getElementById("signing-window").contentWindow;
    const message = event.data;
    if (message.command === "SendParameters") {
      const params = { command: 'parameters', content: signingPayload };
      win.postMessage(params, '*');
    }

    if (message.command === "signedDocument" ||
        message.command === "errorResponse" ||
        message.command === "cancelSign") {
      // TODO: Handle result
    }
  }

  if (window.addEventListener) {
    window.addEventListener("message", onSigningClientMessage);
  } else if (window.attachEvent) {
    window.attachEvent("onmessage", onSigningClientMessage);
  }
</script>
```

The JSON payload being exchanged in the messaging protocol has the fields:

Field	Type	Description
command	string	Command name. Defined later in this chapter.
content	any	Command-specific value, either JSON or a string.

EXAMPLE:

```
{
    command: "parameters",
    content: { "signatureParameters":"eyJ4NWM..", "dtbs":"JVBER.."}
}
```

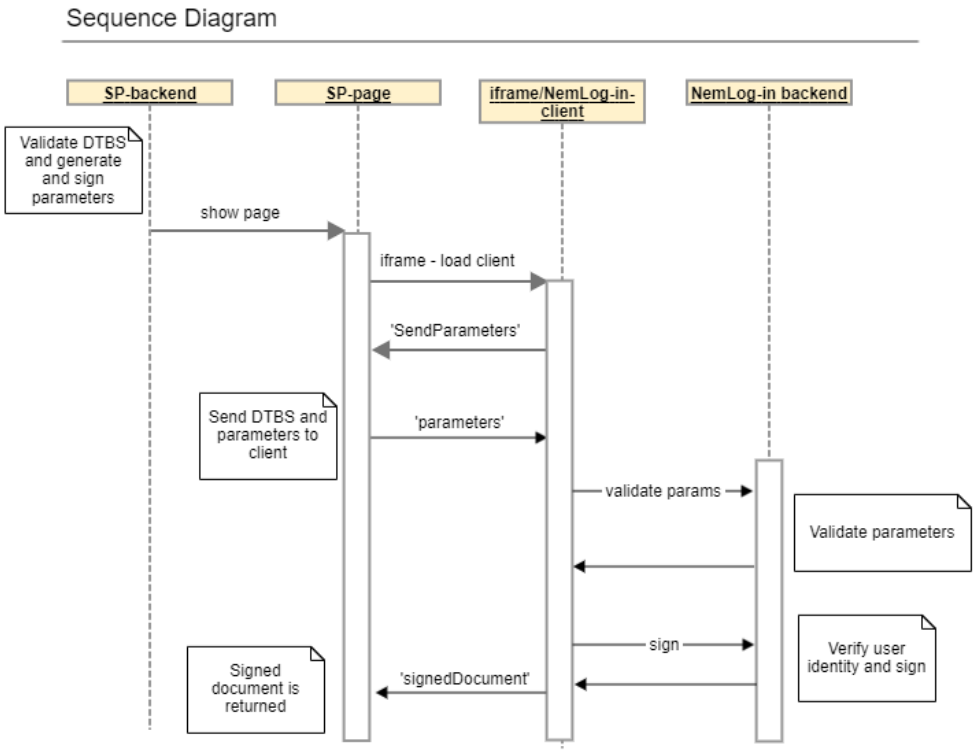


The NemLog-in Signing Client is initialized by taking the following steps

1. The Service Provider web application initializes with an iframe pointing to the Signing Client URL.
2. The Signing Client transmits a 'SendParameters' command to the hosting Service Provider page, to indicate that it is ready to receive the Signing Payload.
3. The Service Provider page transmits a 'parameters' command with the signing payload as content.
4. The Signing Client will then handle the signing of the document with no involvement from the SP web application:
   a. The Signing Client validates the Signing Payload and calls the Signing Backend for validation of the Signature Parameters. This involves verifying the Service Provider's VOCES or FOCES certificate used for JWS-sealing the parameters. If the verification is successful, the Signing Client displays the document to be signed.
   b. Once the Signer has read the document, and has scrolled to the end, the 'Proceed to sign'-button is enabled.
   c. Pressing 'Proceed to sign', the Signer is presented with the NemLog-in Log-in component, where the Signer selects authentication means and continues to provide authentication credenteals. The Signer also selects which electronic identity (person, employee, organization) shall be used for signing.
   d. Upon successful authentication, the log-in window closes, and the Signing Client proceeds to sign the document using AdES LTA (see Appendix Appendix A). The actual signing process involves several calls to the Signing Backend, but at no point of time does the actual document being signed leave the Signing Client.

5. The Signing Client send the 'signedDocument' command to the Service Provide page with the Base64-encoded signed document as content.
6. If the Signer cancel the signing, a 'cancelSign' command is sent to the Service Provider page.
7. If an error occurs during the validation or signing process, an 'errorResponse' command with the error details as content is returned to the Service Provider page.

No matter which way the signing process concludes (signed document, cancellation, or error), the SP web application must subsequently process the returned data and close the Signing Client iframe.



5.4 SP client – NemLog-in Signing Client commands

5.4.1 Signing Client sending 'SendParameters' command

When the Signing Client has loaded and is properly initialized, it is ready to receive the Signing Payload. It sends the 'SendParameters' command to the parent SP page.

Command	Content
SendParameters	empty

EXAMPLE:

```
{
```

```
    command: "SendParameters",
    content: ""
}
```

5.4.2 SP parent send payload to Signing Client

After initializing the Signing Client iframe, the SP parent page must wait for the 'SendParameters' command from the Signing Client. When this is received, the SP parent page must send the Signing Payload.

Command	Content
'parameters'	signing payload (SigningPayloadDTO)

EXAMPLE:

```
{
    command: "parameters",
    content: {"signatureParameters":"eyJ4NWM..", "dtbs":"JVBER.."}
}
```

5.4.3 An error occurs during the signing process

After the Signing Client has received the command 'parameters' with the Signing Payload as content, it will start by validating the document and SP VOCES or FOCES certificate used for sealing the Signature Parameters; then prompt the Signer to authenticate via the NemLog-In log-in component; and then sign the document. If an error occurs during this flow, an 'errorResponse' command is sent back to the SP parent page with Base64-encoded error details in the content. Please see section 3.7 for the `SigningClientError` data model and Appendix B for the list of error codes.

Command	Content
errorResponse	Base64 encoded json (SigningClientError)

EXAMPLE:

```
{
    command: "errorResponse",
    content: "eyJ4NWM.."
}
```

5.4.4 The Signer cancels the signing

After the document has been presented to the Signer, she can proceed with signing or cancel it. If the Signer cancels, a 'cancelSign' command is sent back to the SP parent page.

Command	Content

cancelSign	empty

EXAMPLE:

```
{
    command: "cancelSign",
    content: null
}
```

5.4.5 After successful signing the Signing Client returns the signed document

If the Signing Client successfully signs the document, the signed document is returned to the SP parent page by sending the 'signedDocument' command with the Base64-encoded signed document as the content.

When the document has been signed, the SP must verify that the document has been signed by the expected end user, before presenting the signed document for the end user.

Command	Content
signedDocument	Base64-encoded PAdES or XAdES document

EXAMPLE:

```
{
    command: "signedDocument",
    content: "eyJ4NWM.."
}
```

6 Security Guidelines

This chapter collects general advice and best-practice about securing a Service Provider web site.

6.1 HTTP Headers

It is recommended that Service Providers look into the following HTTP headers and evaluate each carefully regarding its usefulness for the Service Providers application.

The "X-Frame-Options" enables a web page to control whether other pages are allowed to include that web page in an iframe. Including the target page in an iframe is a common approach for certain attacks, and should be disallowed unless specifically needed.

https://developer.mozilla.org/en-US/docs/HTTP/X-Frame-Options

http://tools.ietf.org/html/rfc7034

Specifying the "X-Content-Security-Policy" provides a way to communicate content restrictions to the browser, e.g. that all scripts must come from a specific source or that inline JavaScript should be prohibited.

Setting an appropriate content security policy reduces the impact of rogue content that is injected into a page by software installed on the user's PC.

http://www.html5rocks.com/en/tutorials/security/content-securitypolicy/

http://www.w3.org/TR/CSP/

The "Strict-Transport-Security" header instructs the browser to only access the domain using HTTPS. All unencrypted connections will be redirected to the HTTPS version of the site.

The header can help against the so-called "downgrade attack", where a man-in-the-middle attacker redirects a user to the unencrypted version of a website to eavesdrop or tamper.

The header is obviously superfluous at web sites that are only accessible through HTTPS.

https://developer.mozilla.org/enUS/docs/Security/HTTP_Strict_Transport_Security

http://tools.ietf.org/html/rfc6797

The Open Web Application Security Project (OWASP) lists a few more headers which should also be considered and evaluated.

https://www.owasp.org/index.php/List_of_useful_HTTP_headers

Appendix A. AdES Signature Formats

The signature format created by the NemLog-in Signing Component is aimed for long term archival (LTA), which provides enough data to validate the signature for a period after the signature has been generated.

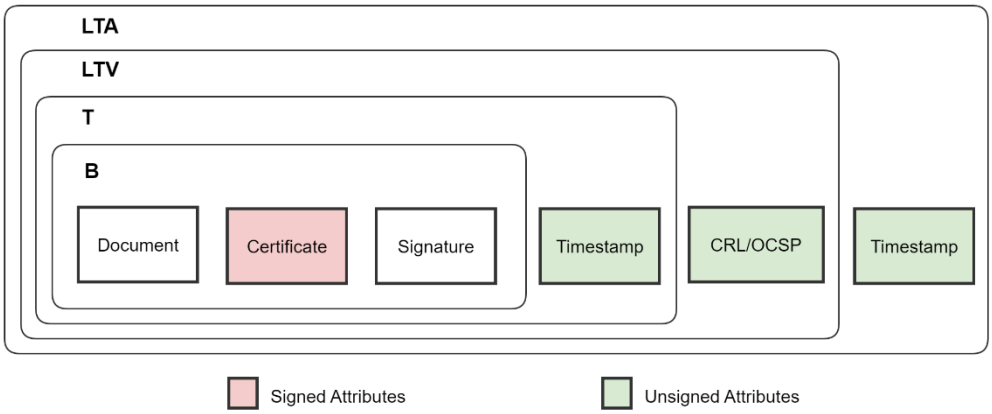
The full specification of the AdES Signature Profile adopted for NemLog-In is defined in [AdES Signature Profile]. This Appendix provides an extract.

The PDF Advanced digital Electronic Signature PAdES [PADES] and XML Advanced digital Electronic Signature XAdES [XADES] are categorised in four groups, each adding on top of the others, specific data to the signature to create a signature format, which meets a posed requirement.

In increasing order of complexity and data, the four groups are:

Group	Data	Security properties
Basic (B)	Original document Signed attributes (incl. signing certificate) Signature	The signature can be used to prove that the identity in the signing certificate has produced a signature over the original document.
Time Stamp (T)	Basic (B) Time Stamp Token	The addition of the Time Stamp Token can be used to prove that the signature existed at the time specified in the token.
Long Term Validation (LTV)	Time Stamp (T) Revocation Information	The inclusion of Revocation Information proves that the signing certificate was not revoked at the time indicated in the information.
Long Term Archival (LTA)	Long Term Validation (LTV) Time Stamp Token	The Time Stamp Token proves that the Revocation Information was available at the time specified in the token.

The illustration below, describes how the classes are related.



PAdES

The PDF-based PAdES LTA format produced by the NemLog-In Signing Component consists of the elements:

- Original PDF/A to be signed
  - It will be extended with a Signature Directory and Document Security Store
- CMS object which contains the basic signature and signature time stamp token, which forms the PAdES-T signature
- A Signature Dictionary containing data including a CMS object
- Document Security Store Dictionary containing data that extends the basic signature to LTA.
  - Extending PAdES-T to PAdES-LTV
- Document Time-Stamp Dictionary
  - Extending PAdES-LTV to PAdES-LTA

For more details, please refer to the [AdES Signature Profile] and the [PAdES] specification.

XAdES

A XML-based XAdES LTA format produced by the NemLog-In Signing Component contains the same semantic information as a PAdES.

With XM signatures, the data to be signed (DTBS) and the signature can be placed relative to each other in the following ways, seen from the signature:

- Enveloped. The signature is included in the DTBS
- Enveloping. The signature contains the DTBS
- Detached. The DTBS and signature are two separate files.

The XML signatures produced by the NemLog-in3 signature service are enveloped.

For more details, please refer to the [AdES Signature Profile] and the [XAdES] specification.

NemLog-In XAdES XSD

The XSD defined for the NemLog-In XAdES signed document format is redproduced below (and included in the SignSDK). The imported XML-DSig XSD is a standard schema file and not included here.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://dk.gov.certifikat/nemlogin/v0.0.1#"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            targetNamespace="http://dk.gov.certifikat/nemlogin/v0.0.1#"
            elementFormDefault="qualified">
    <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
                schemaLocation="xmldsig-core-schema.xsd"/>

    <!-- Start SignedDocument -->
    <xsd:element name="SignedDocument" type="SignedDocumentType"/>
    <xsd:complexType name="SignedDocumentType">
        <xsd:sequence minOccurs="1">
            <xsd:element name="SignText" type="SignTextType"/>
            <xsd:element ref="ds:Signature"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="SignText" type="SignTextType"/>
    <xsd:complexType name="SignTextType">
        <xsd:sequence>
            <xsd:choice maxOccurs="1">
                <xsd:element name="PlainText" type="PlainTextType"/>
                <xsd:element name="HTMLDocument" type="HTMLDocumentType"/>
                <xsd:element name="PDFDocument" type="PDFDocumentType"/>
                <xsd:element name="XMLDocument" type="XMLDocumentType"/>
            </xsd:choice>
            <xsd:element name="Properties" type="PropertiesType" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="PlainTextType">
        <xsd:sequence minOccurs="1">
            <xsd:element name="Document" type="xsd:base64Binary"/>
```

```
        <xsd:element name="Rendering" type="RenderingType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="HTMLDocumentType">
    <xsd:sequence minOccurs="1">
        <xsd:element name="Document" type="xsd:base64Binary"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PDFDocumentType">
    <xsd:sequence minOccurs="1">
        <xsd:element name="Document" type="xsd:base64Binary"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="XMLDocumentType">
    <xsd:sequence minOccurs="1">
        <xsd:element name="Document" type="xsd:base64Binary"/>
        <xsd:element name="Transformation" type="xsd:base64Binary"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RenderingType">
    <xsd:sequence minOccurs="1">
        <xsd:element name="UseMonoSpaceFont" type="xsd:boolean"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PropertiesType">
    <xsd:sequence>
            <xsd:element name="Property" type="PropertyType" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PropertyType">
    <xsd:sequence>
        <xsd:element name="Key" type="xsd:string"/>
```

```
            <xsd:choice>
                <xsd:element name="StringValue" type="xsd:string"/>
                <xsd:element name="BinaryValue" type="xsd:base64Binary"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Appendix B. Error codes

Signing API Error Codes (Signing Component Backend)

Common Signing Component error codes

CMN001	Internal Signing Component error
CMN002	Invalid Sign Flow session
CMN003	Unauthorized Access
CMN004	Invalid Signing API Endpoint
CMN005	Connection timeout
CMN006	Missing or invalid correlation ID in request

Error codes pertaining to the sign-flow

FLW001	Internal sign-flow error
FLW002	Missing or invalid sign-flow session ID
FLW003	Invalid sign-flow step

Error codes pertaining to the BeginSignFlow validation API endpoint

SPV001	Error parsing Signature Parameters as JWS
SPV002	Invalid certificate included in Signature Parameters JWS
SPV003	Invalid Signature Parameters JWS signature
SPV004	Missing Signature Parameters mandatory field
SPV005	Invalid Signature Parameters field type
SPV006	Invalid Signature Parameters field value
SPV007	Unknown Signature Parameters field
SPV008	Invalid Signature Parameters
SPV009	Missing parameter for Service Provider flow
SPV010	Invalid parameter for Broker flow
SPV011	Invalid Signature Parameter entityID

Error codes pertaining to document signing

SIGN001	Invalid Signature Parameters field dtbsSignedInfo
SIGN002	General Document Signing error
SIGN003	Error issuing certificate
SIGN004	Revocation check failed
SIGN005	Timestamp generation failed
SIGN006	Error creating PAdES LTV Signature
SIGN007	Error creating PAdES LTA Signature
SIGN008	Internal error timeout caCertsFromGuuid
SIGN009	Unable to create SAD

SAML errors whilst logging in during a signing flow

SAML001	Error generating SAML SP metadata
SAML002	Error generating SAML AuthnRequest
SAML004	Error awaiting SAML login
SAML020	SAML Response error - error parsing SSO SAML Response
SAML021	SAML Response error - invalid InResponseTo
SAML022	SAML Response error - invalid destination
SAML023	SAML Response error - invalid lifetime
SAML024	SAML Response error - unsuccessful response status

SAML025	SAML Response error - contains DTD
SAML026	SAML Response error - invalid Issuer
SAML027	SAML Response error - invalid structure
SAML028	SAML Response error - error serializing SAML assertion
SAML029	SAML Response error - invalid flow type
SAML040	SAML Assertion error - error parsing SAML assertion
SAML041	SAML Assertion error - signature error
SAML042	SAML Assertion error - decryption error
SAML043	SAML Assertion error - missing or invalid attribute
SAML044	SAML Assertion error - missing or invalid Subject
SAML045	SAML Assertion error - missing or invalid Audience
SAML046	SAML Assertion error - AuthnStatement not current
SAML047	SAML Assertion error - invalid Issuer
SAML048	SAML Assertion error - invalid structure

Signing Client Error Codes

SCE010	Invalid Signature Parameters document format
SCE011	Unknown error
SCE012	Unknown Signature Format
SCE013	Invalid DTBS digest algorithm
SCE014	Invalid DTBS digest
SCE015	Invalid SignedInfo Digest
SCE016	Invalid DTBS Format
SCE017	Document to be signed exceeds max size
SCE018	Invalid Flow Type
SCE019	Internal Error in initializing Signed SDK
SCE020	Internal Error in initializing Signer Creation Key
SCE021	Invalid browser

SignSDK Error Codes

SDK001	Error loading SD
SDK002	Invalid Signature Parameters
SDK003	Service Implementation unavailable
SDK004	Error JWS-signing Signing Payload
SDK005	Error generating DTBS signature template
SDK006	Error computing DTBS digest
SDK007	Error transforming SD to PDF DTBS document
SDK008	Error adding attachments to PDF DTBS document
SDK009	Error transforming SD to XML DTBS document
SDK010	Error validating SD
SDK011	Error validating document signature

Appendix C. HTML Whitelists

HTML Element Whitelist

The allowed HTML elements and attributes for Signer's Documents of type HTML are listed below:

HTML Element	Supported attributes
html	Xmlns
body	text bgcolor class style
head	
style	Type
title	
p	align bgcolor style class
div span	align bgcolor style class
ul	style class
ol	start type style class
li	class style
h1 h2 h3	class style
h4 h5 h6	class style
meta	charset http-equiv name content
table	border cellspacing cellpadding width align
tr	bgcolor class style
th td	bgcolor rowspan colspan align valign width class style
i b u	
center	
a	href name
tbody thead tfoot	
br	

Links may point to named anchors within the document but cannot point to external documents.

CSS Whitelist

External CSS files are not supported, and only alimited set subset of CSS styles are supported.

background	background-color	bottom	color
clear	display	float	height
left	line-height	overflow	position
right	top	width	white-space
margin*	padding*	border*	font*
text*	list*		

CSS styles tagged with a * above indicate entire font families, so e.g. "margin*" includes "margin", "margin-top", "margin-bottom", etc.

Background is not allowed include an image, as external URLs are not supported.

Appendix D. PDF Whitelists

Base PDF Fonts

Fonts in the PDF Signer's Document should be embedded in the document. However, the following standard PDF fonts need not be embedded:

- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Symbol
- ZapfDingbats

PDF Whitelist

The following sections are the complete PDF whitelists for Signer's Documents of type PDF

WHITELISTED TYPES

/FontDescriptor	/Font	/Metadata

WHITELISTED KEYS

/Encoding	/XObject	/Dests
/ExtGState	/ProcSet	/Dest
/ColorSpace	/Properties	/Info
/Pattern	/BaseFont	/Font
/Shading	/Name	/Differences

WHITELISTED NAMES

/1.1	/1.2	/1.3

/1.4
/1.5
/1.6
/1.7
/2.2
/83pv-RKSJ-H
/90ms-RKSJ-H
/90ms-RKSJ-V
/90msp-RKSJ-H
/90msp-RKSJ-V
/90pv-RKSJ-H
/A
/A85
/AC
/ADBE
/AESV2
/AHx
/AIS
/AN
/AP
/AS
/ASCII85Decode
/ASCIIHexDecode
/AbsoluteColorimetric
/Accepted
/AccurateScreens
/Action
/ActualText
/Add-RKSJ-H
/Add-RKSJ-V
/AddRevInfo
/Adobe.PPKLite
/After
/All
/AllOff
/AllOn
/AllPages
/Alpha
/AlphaNum
/Alphabetic
/Alt
/Alternate
/AlternateImages
/AlternatePresentations
/Alternates
/Angle
/Annot
/AnnotStates
/Annotation
/Annotations
/Annots
/AntiAlias
/AnyOff
/AnyOn
/App
/AppDefault
/Approced
/Art
/ArtBox
/AsIs
/Ascent
/Attached
/Attestation
/AuthEvent
/Author
/Auto
/AvgWidth
/B
/B5pc-H
/B5pc-V
/BBox
/BC
/BE
/BG
/BG-EUC-H
/BG-EUC-V
/BG2
/BM
/BS
/Background
/BackgroundColor
/BarcodePlaintext
/BaseEncoding
/BaseFont

/BaseState
/BaseVersion
/BaselineShift
/Bead
/Before
/BibEntry
/BitsPerComponent
/BitsPerCoordinate
/BitsPerFlag
/BitsPerSample
/Black
/BlackPoint
/BlackIs1
/BleedBox
/Block
/BlockAlign
/BlockQuote
/Blue
/Border
/BorderColor
/BorderStyle
/BorderThickness
/Both
/Bounds
/BoxColorInfo
/ByteRange
/C
/C0
/C1
/CA
/CCF
/CCITTFaxDecode
/CF
/CFM
/CICI.SignIt
/CIDFontType0
/CIDFontType0C
/CIDFontType2
/CIDInit
/CIDSet
/CIDSystemInfo
/CIDToGIDMap
/CMap
/CMapName
/CMapType
/CNS-EUC-H
/CNS-EUR-V
/CO
/CP
/CS
/CYX
/CalGray
/CalRGB
/Cancelled
/Cap
/CapHeight
/Caption
/Caret
/Catalog
/Center
/CenterWindow
/Cert
/Changes
/CharProcs
/CharSet
/Chart
/Chartsheet
/Circle
/ClassMap
/Code
/ColSpan
/Collection
/CollectionField
/CollectionItem
/CollectionSort
/CollectionSubItem
/Color
/ColorBurn
/ColorDodge
/ColorSpace
/ColorTransform
/Colorants
/Colors
/Column

/ColumnCount
/ColumnGap
/ColumnWidth
/Columns
/Comment
/Completed
/Components
/Confidential
/Configs
/ContactInfo
/Content
/Contents
/Coords
/Copy
/CosineDot
/Count
/Courier
/Courier-Bold
/Courier-BoldOblique
/Courier-Oblique
/Create
/CreationDate
/Creator
/CreatorInfo
/CropBox
/Cross
/Crypt
/CryptFilter
/CryptFilterDecodeParms
/Cyan
/D
/DA
/DCTDecode
/DL
/DTC
/DW
/DW2
/DamagedRowsBeforeError
/Darken
/Dashed
/Data
/Date
/Decimal
/Decode
/DecodeParams
/DecodeParms
/Default
/DefaultForPrinting
/Delete
/Departmental
/Desc
/DescendantFonts
/Descent
/Design
/Dest
/DestOutputProfile
/Dests
/DevDepGS_BG
/DevDepGS_FL
/DevDepGS_HT
/DevDepGS_OP
/DevDepGS_TR
/DevDepGS_UCR
/DeveloperExtensions
/DeviceCMY
/DeviceCMYK
/DeviceColorant
/DeviceGray
/DeviceN
/DeviceRGB
/DeviceRGBK
/Diagram
/Diamond
/Dialogsheet
/Difference
/Differences
/DigestLocation
/DigestMethod
/DigestValue
/Dingbats
/DingbatsRot
/Direction
/Disc
/DisplayDocTitle

/Distribute
/Div
/DocMDP
/DocOpen
/Document
/Domain
/DotGain
/Dotted
/Double
/DoubleDot
/Draft
/Duplex
/DuplexFlipLongEdge
/DuplexFlipShortEdge
/E
/EF
/EFF
/EFOpen
/ETen-B5-H
/ETen-B5-V
/ETenms-B5-H
/ETenms-B5-V
/EUC-H
/EUC-V
/EarlyChange
/Ellipse
/EllipseA
/EllipseB
/EllipseC
/Encode
/EncodedByteAlign
/Encoding
/Encrypt
/EncryptMetadata
/End
/EndIndent
/EndOfBlock
/EndOfLine
/Endnote
/EntcryptMetaData
/Entrust.PPKEF
/ExData
/Exclude
/Exclusion
/Experimental
/Expired
/Export
/ExportState
/Ext-RKSJ-H
/Ext-RKSJ-V
/ExtGState
/Extend
/Extends
/ExtensionLevel
/Extensions
/ExternalOPIdicts
/ExternalRefXobjects
/ExternalStreams
/F
/F9+0
/FD
/FG
/Figure
/FL
/False
/Ff
/FieldMDP
/Fields
/FillIn
/Filter
/Final
/First
/FirstChar
/FirstPage
/Fit
/FitB
/FitBH
/FitBV
/FitH
/FitR
/FitV
/FitWindow
/FixedPrint
/Fl

/Flags
/FlatDecode
/FlateDecode
/Font
/FontBBox
/FontDescriptor
/FontFamily
/FontFauxing
/FontFile
/FontFile2
/FontFile3
/FontMatrix
/FontName
/FontStretch
/FontWeight
/Footer
/Footnote
/ForComment
/ForPublicRelease
/Form
/FormEx
/FormType
/Formula
/FreeText
/Frequency
/FullSave
/FullScreen
/Function
/FunctionType
/Functions
/G
/GBK-EUC-H
/GBK-EUC-V
/GBK2K-H
/GBK2K-V
/GBKp-EUC-H
/GBpc-EUC-H
/GBpc-EUC-V
/GTS_PDFA1
/GTS_PDFX
/Gamma
/Generic
/GenericRot
/GlyphOrientationVertical
/GoTo
/GoToRemoveActions
/Gray
/Green
/Groove
/Group
/H
/H1
/H2
/H3
/H4
/H5
/H6
/HF
/HKana
/HKanaRot
/HKscs-B5-H
/HKscs-B5-V
/HRoman
/HRomanRot
/HT
/Halftone
/HalftoneName
/HalftoneType
/Hanzi
/HardLight
/Header
/Headers
/Height
/Height2
/Help
/Helvetica
/Helvetica-Bold
/Helvetica-BoldOblique
/Helvetica-Oblique
/Hidden
/HideAnnotationActions
/HideMenubar
/HideToolbar
/HideWindowsUI

/Highlight
/HojoKanji
/Hue
/I
/IC
/ICCBased
/ID
/IDS
/IDTree
/IF
/IRT
/IT
/IX
/Identify
/Identify-H
/Identify-V
/Image
/ImageB
/ImageC
/ImageI
/ImageMask
/Import
/Include
/Ind
/Index
/Indexed
/Info
/Ink
/InkList
/Inline
/InlineAlign
/InlineShape
/Insert
/Inset
/Intent
/InterPolate
/Interpolate
/InvertedDouble
/InvertedDoubleDot
/InvertedEllipseA
/InvertedEllipseC
/InvertedSimpleDot
/Invisible
/Issuer
/ItalicAngle
/JBIG2Decode
/JBIG2Globals
/JPXDecode
/JavaScriptActions
/Justify
/K
/KSC-EUC-H
/KSC-EUC-V
/KSCms-UHC-H
/KSCms-UHC-HW-H
/KSCms-UHC-HW-V
/KSCms-UHC-V
/KSCpc-EUC-H
/Kana
/Kanji
/Key
/KeyUsage
/Keywords
/Kids
/L
/L2R
/LBody
/LC
/LE
/LI
/LJ
/LL
/LLE
/LLO
/LW
/LWZDecode
/Lab
/Lang
/Language
/Last
/LastChar
/LastModified
/LastPage
/LaunchActions

/Layout
/Lbl
/Leading
/Legal
/LegalAttestation
/Length
/Length1
/Length2
/Length3
/Level1
/Lighten
/Limits
/Line
/LineHeight
/LineThrough
/LineX
/LineY
/Linearized
/ListMode
/ListNumbering
/Location
/Lock
/Locked
/LockedContent
/LowerAlpha
/LowerRoman
/LrTb
/Luminosity
/M
/MCID
/MCR
/MDP
/MK
/ML
/MMType1
/MN
/MacExpertEncoding
/MacRomanEncoding
/Macrosheet
/Magenta
/MarkInfo
/MarkStyle
/Marked
/Mask
/Matrix
/Matte
/MaxWidth
/Maxtrix
/Measure
/MediaBox
/Metadata
/Middle
/MissingWidth
/MixingHints
/ModDate
/Modify
/MovieActions
/Msg
/Multiply
/N
/NChannel
/NM
/Name
/Named
/Names
/NeedsRendering
/NewParagraph
/Next
/NextPage
/NoRotate
/NoView
/NoZoom
/NonEFontNoWarn
/NonEmbeddedFonts
/NonFullScreenPageMode
/NonStruct
/None
/Normal
/NotApproced
/NotForPublicRelease
/Note
/NumCopies
/NumberFormat
/Nums

/O
/OBJR
/OC
/OCG
/OCGs
/OCMD
/OCProperties
/OFF
/OID
/ON
/OP
/OPI
/OPM
/OS
/Obj
/ObjStm
/OneColumn
/Online
/Open
/OpenType
/OptionalContent
/Order
/Ordering
/Org
/Outlines
/OutputCondition
/OutputConditionIdentifier
/OutputIntent
/OutputIntents
/Outset
/Overlay
/Overline
/P
/PCM
/PDF
/PS
/PZ
/Padding
/Page
/PageElement
/PageLabel
/PageLabels
/PageLayout
/PageMode
/Pages
/Pagination
/PaintType
/Paragraph
/Parent
/ParentTree
/ParentTreeNextKey
/Part
/Pattern
/PatternType
/Perceptual
/Perms
/Pg
/PickTrayByPDFSize
/PieceInfo
/Placement
/PolyLine
/PolyLineDimension
/Polygon
/PolygonCloud
/PolygonDimension
/Popup
/PreRelease
/Predictor
/Preferred
/PresSteps
/PreserveRB
/Prev
/PrevPage
/Preview
/Print
/PrintArea
/PrintClip
/PrintPageRange
/PrintScaling
/PrinterMark
/PrintersMarks
/PrintingOrder
/Private
/ProcSet

/Process
/Producer
/Prop_AuthTime
/Prop_AuthType
/Prop_Build
/Properties
/Proportional
/ProportionalRot
/PubSec
/Q
/QuadPoints
/Quote
/R
/R2L
/RBGroups
/RC
/RD
/REx
/RI
/RIPEMD160
/RL
/RT
/Range
/ReadOnly
/Reason
/Reasons
/Receipients
/Rect
/Red
/Redition
/Ref
/Reference
/Registry
/RegistryName
/Rejected
/RelativeColorimetric
/Rendition
/Renditions
/Requirements
/Resources
/Rhombold
/Ridge
/RlTb
/Role
/RoleMap
/Root
/Rotate
/Round
/Row
/RowSpan
/Rows
/Ruby
/RubyAlign
/RubyPosition
/RunLengthDecode
/S
/SA
/SE
/SHA1
/SHA256
/SHA384
/SHA512
/SM
/SMask
/SMaskInData
/SS
/SV
/SVCert
/Saturation
/Schema
/Scope
/Screen
/Sect
/Separation
/SeparationColorNames
/SeparationInfo
/SetOCGState
/Shading
/ShadingType
/Sig
/SigFieldLock
/SigQ
/SigRef
/Signature

/SimpleDot
/Simplex
/SinglePage
/Size
/Slide
/SoftLight
/Sold
/Solid
/Solidities
/Sort
/SoundActions
/SpaceAfter
/SpaceBefore
/Span
/SpawnTemplate
/SpotFunction
/Square
/Squiggly
/St
/Stamp
/Standard
/Start
/State
/StartIndent
/StemH
/StemV
/Stm
/StmF
/StmOwn
/StrF
/StrikeOut
/StructElem
/StructParent
/StructParents
/StructTreeRoot
/Style
/SubFilter
/SubType
/Subj
/Subject
/SubjectDN
/SubmitStandalone
/Subtype
/Summary
/SummaryView
/Supplement
/Suspects
/Sy
/Symbol
/T
/TBody
/TBorderStyle
/TD
/Textbox
/TFoot
/TH
/THead
/TK
/TOC
/TOCI
/TP
/TPadding
/TR
/TR2
/Table
/Tabs
/TbRl
/TemplateInstantianted
/Templates
/Text
/TextAlign
/TextDecorationColor
/TextDecorationThickness
/TextDecorationType
/TextIndent
/Thread
/Threads
/Thumb
/TilingType
/TimeStamp
/Times-Bold
/Times-BoldItalic
/Times-Italic
/Times-Roman

/Title
/ToUnicode
/Toggle
/ToggleNoView
/Top
/TopSecret
/Trans
/TransferFunction
/TransformMethod
/TransformParams
/Transparency
/TrapNet
/TrapRegions
/TrapStyles
/Trapped
/Trapping
/TrimBox
/True
/TrueType
/TrueTypeFonts
/TrustedMode
/Ttl
/TwoColumnLeft
/TwoColumnRight
/TwoPageLeft
/TwoPageRight
/Type
/Type0
/Type1
/Type1C
/Type3
/U
/UCR
/UCR2
/UR
/UR3
/URIActions
/Unchanged
/Underline
/UniCNS-UCS2-H
/UniCNS-UCS2-V
/UniCNS-UTF16-H
/UniCNS-UTF16-V
/UniGB-UCS2-H
/UniGB-UCS2-V
/UniGB-UTF16-H
/UniGB-UTF16-V
/UniJIS-UCS2-H
/UniJIS-UCS2-HW-H
/UniJIS-UCS2-HW-V
/UniJIS-UCS2-V
/UniJIS-UTF16-H
/UniJIS-UTF16-V
/UniKS-UCS2-H
/UniKS-UCS2-V
/UniKS-UTF16-H
/UniKS-UTF16-V
/Unknown
/Unmarked
/UpperAlpha
/UpperRoman
/Usage
/UseAttachments
/UseCMap
/UseNone
/UseOC
/UseOutlines
/UseThumbs
/User
/UserProperties
/UserUnit
/V
/V2
/VE
/VP
/VeriSign.PPKVS
/Version
/Vertices
/VerticesPerRow
/View
/ViewArea
/ViewClip
/ViewState
/ViewerPreferences

/Viewport	/Ysquare
/VisiblePages	/ZapfDingbats
/W	/Zoom
/W2	/adbe.pkcs7.detached
/WMode	/adbe.pkcs7.sha1
/Warichu	/adbe.x509.rsa_sha1
/Watermark	/ca
/WhitePoint	/cb
/Width	/checked
/Width2	/max
/Widths	/min
/WinAnsiEncoding	/neutral
/Workbook	/null
/Worksheet	/off
/WritingMode	/on
/X	/op
/XFAResources	/pb
/XHeight	/rb
/XML	/tv
/XObject	/Artifact
/XRef	/Link
/XRefStm	/F1, F2, … /FX
/XStep	
/XYZ	
/Xsquare	
/Y	
/YStep	
/Yellow	