Table of Contents

1 Introduction

This document describes the TU Services platform. How to setup the platform. How to perform signing with the platform. And the options available with regards to signing.

The TU services platform supports MitID and NemLog-in3 signing infrastructures.

This is done through either the Document API (described in section 3) for both the MitID and NemLog-in3 infrastructures.

Alternatively, a lightweight standalone NemLog-in3 integration is also supported. (described in section 4)

The primary use of the Document API is to support multiple signers on the same document.

It can be used to setup a PDF or text for signing by multiple required signers and supports validation properties like CPR and specific MitID UUIDs.

For private customers, NemLog-in3 validation will require a prior login to the NemLog-in3 platform. This is necessary to obtain sufficient information on the pending signer. This is necessary to verify the identity of the signer subsequently.

The Document API also allows for using both MitID and NemLog-in3 signatures on the same document. Allowing for flexible solutions.

For users wishing to use a simpler model for NemLog-in3 signing. The TU services platform also supports this. In the simple flow, the signing via NemLog-in3 can be done without using the Document API. In this case, the signed PDF will be available in the session for the user to see and the system to store.

The following subsections is a short description of respectively the MitID and NemLog-in3 signature formats, infrastructure and challenges.

1.1 NemLog-in3 signatures

Signing a document (PDF, text) using NemLog-in3 results in a PAdES created wrapping the original document. This results in the same issues as with old NemID signatures.

The TU Services Document API can be used in the same way for NemLog-in3 signatures. The same system has been extended such that it can also manage NemLog-in3 PAdES signatures.

For private customers. NemLog-in3 only allows specific usages of their lookup services. This is problematic, as it then is not possible to validate the identity of the signer. The reason for this is that with MitID the Nemlog-in3 signing solution will be based on the NemLog-in3 CPR UUID identifier and not the generic MitID UUID identifier commonly used by normal private service providers.

The current solution to this problem is to require, that the user prior to signing uses the NemLog-in3 infrastructure to login. This will supply the system with enough information to be able to validate the identity of the pending signer.

1.2 MitID acceptances

It it also possible to use MitID without going through NemLog-in3. This can be done using Nets eID Broker.

The Nets eID Broker signature option functions as an authentication-based-signing flow, where the user, instead of signing the documents, uses their MitID to authenticate, that they have seen and verified the contents presented during the signing process.

The TU Services Document API can be used to request this type of acceptances.

This flow results in a transaction token describing the signature, and an ocsp response, which can be used to check revocation info on said signature.

Transaction token and ocsp response are embedded in the final signature file from the Document API

2 TU-services Setup

To setup TU-services for document signing. A SPS.Client is required.

2.1 .NET

The SPS.Client consist of a single.NET DLL (SPS.Client.dll). This and its dependencies should be referenced in your system. (SPS.Core.dll, Newtonsoft.Json 11.0.0.0).

An assembly binding might be necessary, if there a conflicts between used versions of Newtonsoft.Json:

```
<runtime>
 <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
  <dependentAssembly>
   <assemblyIdentity name="Newtonsoft.Json" culture="neutral" publicKeyToken="30ad4fe6b2a6aeed" />
   <bindingRedirect oldVersion="0.0.0.0-11.0.0.0" newVersion="11.0.0.0" />
  </dependentAssembly>
 </assemblyBinding>
</runtime>
```

A section of configuration must be supplied to properly setup SPS.Client. It is up to the system to decide how this is done.

We recommend using a web.config configuration section as the following:

```
<configuration>
 <configSections>
  <section name="SPS.Client"
  type="SPS.Client.Configuration.SpsClientConfigurationSection, SPS.Client" />
  ...

<SPS.Client CredentialsJsonFile="~/App_Data/demo-credentials.json" />
```

Note that the json will be supplied by us. (demo-credentials.json in this example)

2.2 Java

The SPS.Client library is also supported as a maven dependency. To use the SPS.Client library in Java, include the following dependency into your pom.xml:

```
<dependency>
    <groupId>dk.signaturgruppen.sps</groupId>
    <artifactId>spsclient-java</artifactId>
</dependency>
```

The maven repo can be found at https://www.signaturgruppen.dk/download/m2. Code examples written in this document should give a good indication of how the system can be used in Java as well. For more examples. Take a look at the JAva demo mentioned in the following section.

3 Demos

Documentation and demos can be found on https://authentication.signaturgruppen.dk

For.NET. We refer to our demo application found at https://authentication.signaturgruppen.dk/Home/Download for a more detailed example of the Document API.

For Java. We refer to our java demo application found at https://authentication.signaturgruppen.dk/Home/Download. This demo contains both login, document and signing integration to TU-services. For the use case of NemLog-in3. Only the documentAPI and signing standalone are of interest.

4 Document API

The document API is a classic backend-to-backend REST API for most parts, implemented by our SPS.Client library. When doing the actual signing, the API works by setting up an iframe, generating a JavaScript block which is placed beside the iframe and communicating with our TU Services backend through the iframe using the JavaScript based HTML 5 Web Messaging API.

We refer to the SPS.Client for a more general technical introcution to the TU Services integration, with more detailed info about error handling etc.

The workflow of the document API is as follows:

1. Create a document with X required signers.
2. Sign the document with the X required signers.
3. Acquire the resulting PADES, where X signatures for each signer is present.

Both MitID and NemLog-in3 is supported for these flows

4.1 Create new document

Example of how to create a new document from a PDF with two required signers.

```
var docGuid = new DocumentApiClient()
          .CreateDocumentWithPdf(toBeSignedPdfBytes)
          .WithRequiredSigner(new RequiredSigner("username1")
               .WithIdentity(SignerIdentity.Name("Foo Bar"))
               .WithIdentity(SignerIdentity.Titel("Something A/S")))
          .WithRequiredSigner(new RequiredSigner("username2")
               .WithIdentity(SignerIdentity.Name("Niels Lauritsen"))
               .WithIdentity(SignerIdentity.Titel("CEO Firma A/S")))
          .DocumentTitle("Demo document " + pdfName)
          .Invoke();
```

4.2 Sign Document

To have user with username username1 sign a previously created document, use the client as follows:

A flowbuilder using NemLog-in3 can be created.

```
var flowBuilder = new NL3DocumentFrameBuilder(docGuid).SignerId(username1);
var scriptBlock = flowBuilder.GetScript("pathToYourValidationHandler");
var frameUrl = flowBuilder.FrameUrl;
```

Or a flowbuilder using MitID:

```
var flowBuilder = new MitIDAcceptDocumentFrameBuilder((Guid) Session["docGuid"]).SignerId(user.Username);
var script = flowBuilder.GetScript(Url.Action("FinishNetsBroker", "Sign"));
var iframeUrl = flowBuilder.FrameUrl;
```

2: Setup the iframe in the desired size using the "frameUrl". It is recommended for the NemLog-in3 frame to be full screen.

3: Insert the JavaScript tag produced in "scriptBlock" on the webpage beside the iframe.

The JavaScript generated by "GetScript()" will communicate with the TU Services backend and setup a MitID/NemLog-in3 signing flow inside the iframe. When the user has signed the document, the result will be posted to "pathToYourValidationHandler" and found in the "result" value of the post parameters.

4: Decrypt and verify the response by using the client using NemLog-in3:

```
var docAuthInfo = new NL3FlowValidator().Validate(result);
```

And using Nets eID Broker:

```
var docAuthInfo = new MitIDAcceptFlowValidator().Validate(result);
```

The.Validate(result); call throws a ValidationException if the result is anything but a successful MitID/NemLog-in3 signature flow. Refer to the SPS.Client documentation for error handling and logging.

4.3 Signing parameters

The signing parameters are used to control parts of the NemLog-in3 document signing. The current possible parameters are:

Parameter	Value	Description
SsnPersistenceLevel < Optional >	Global / Session	Used to specify the persistence level of the uuid returned with the certificate from NemLog-in3. **Global**: Returns the CPR-UUID. This is the **default** option in the TU-services platform. **Session**: Returns a unique UUID. NemLog-in3 EIA lookup services are required to get information on the user with this option.
SignedDocumentFormat < Optional >	PAdES / XAdES	Used to specify the returned format of the signed document. **PAdES**: PDF format. This is the **default** option **XAdES**: XML format *(will be supported in a later version)*

Signing parameters are configured in the following way:

```
    var flowBuilder = new NL3DocumentFrameBuilder(docGuid)
        .WithSSNPersistenceLevel(SSNPersistenceLevel.Global)
```

4.4 Signing result

The result of the document API flow returns the following upon completing a flow. These are the NemLog-in3 document API specific parameters.

Parameter	Description
DocumentId	The Guid used to identify the document in question.

4.5 Extended validation of signer properties

If additional validation is needed of signer properties like danish CPR, required UUIDs. Validation can be specified for NemLog-in3 signing:

```
var flowBuilder = new NL3DocumentFrameBuilder(docGuid)
    .SignerId(username1)
    .SignerPid("PID") <- optional
    .SignerUUID("UUID") <- optional
```

If specified, the required PID or UUID will be verified against the NemLog-in3 signature. Note that a prior NemLog-in3 login is required to be able to verify the required attributes.

And for MitID acceptance:

```
var flowBuilder = new MitIdAcceptDocumentFrameBuilder(docGuid)
```

```
        .SignerId(username1)
        .SignerPid("PID") <- optional
        .SignerUUID("UUID") <- optional
        .SignerUUID("CPR") <- optional
```

note that this UUID id different from the one used in NemLog-in3. This is the MitID UUID.

4.6 Get document summary

To retrive a lightweight summary of the current state of the document, use

```
var summary = new DocumentApiClient().GetSummary(docGuid);
```

4.7 Get document

At any point the full state of the document can be retrieved

```
var documentData = new DocumentApiClient().GetDocument(docGuid);
```

Where the documentData object contains all constructed MitID and NemLogin signatures and info of signers and the document, as well as the original PDF bytes.

4.8 Delete document

A document can be deleted, by calling

```
new DocumentApiClient().Delete(docGuid);
```

This will mark the document as DELETED in our backend and all sensitive data will be deleted from our database. We still maintain the log needed to verify that the proper validation have been done when creating the signatures.

4.9 Get all

To retrieve a summary-list of all active documents

```
var allDocuments = new DocumentApiClient().GetAll();
```

4.10 Get pending

To retrieve a summary-list of all active documents with required signers without a completed signature

```
var pendingDocuments = new DocumentApiClient().GetPending();
```

4.11 Get signed

To retrieve a summary-list of all active documents where all required signers have signed the document

```
var signed = new DocumentApiClient().GetSigned();
```

4.12 Build PAdES

At any point a PAdES can be constructed which will include a siganture page at the end of the PDF document with all current signers of the document.

```
var padesBytes = new DocumentApiClient().CreatePdfAdvancedElectronicSignature(docGuid);
```

4.13 Automatic cleanup

The platform will automatically delete any documents older than 30 days with no new signatures added in more than 90 days.

5 NemLog-in3 standalone

For system that does not require multiple signers or where the added complexity is unnecessary. The TU-Services platform also supports a simple signature flow.

The workflow of the simple standalone flow is as follows:

1. Supply the system with a base64 string of a valid PDF document.
2. Sign the document with a signer.
3. Store the resulting PAdES from the session of the system.

5.1 Standalone signing

1: A simple NemLog-in3 signature flow can be initiated with the following line. Keeping the file extension in the document title is important!

```
var flowBuilder = new NL3SignFrameBuilder().SignPdf(binData).WithTitle("file.pdf");
var scriptBlock = flowBuilder.GetScript("pathToYourValidationHandler");
var frameUrl = flowBuilder.FrameUrl;
```

with simple text:

```
var flowBuilder = new NL3SignFrameBuilder().SignText("FooBAR").WithTitle("aRandomTitle.txt");
```

with html:

```
var flowBuilder = new NL3SignFrameBuilder().SignHtml(binData).WithTitle("file.html");
```

or with xml:

```
var flowBuilder = new NL3SignFrameBuilder()
      .SignXml(Convert.FromBase64String(xmlB64), Convert.FromBase64String(xslB64))
      .WithTitle("Test.xml");
```

2: Setup the iframe in the desired size using the "frameUrl". It is recommended for the NemLog-in3 frame to be full screen.

3: Insert the JavaScript tag produced in "scriptBlock" on the webpage beside the iframe.

The JavaScript generated by "GetScript()" will communicate with the TU Services backend and setup a NemLog-in3 signing flow inside the iframe. When the user has signed the document, the result will be posted to "pathToYourValidationHandler" and found in the "result" value of the post parameters.

4: Descrypt and verify the reponse by using the client:

```
var flowResult = new NL3FlowValidator().Validate(result);
```

The.Validate(result); call throws a ValidationException if the result is anything but a successful NemLog-in3 signature flow.
Errors can occur before the signing iframe is loaded. In this case, the error is contained within the result, and calling Validate(result) will result in a ValidationException describing the corresponding error.

5: Get the PAdES from the flowresult for storing on your system:

```
var pAdES = flowResult.Result;
//STORE THE PADES
```

Here is a flow-diagram describing the flow:

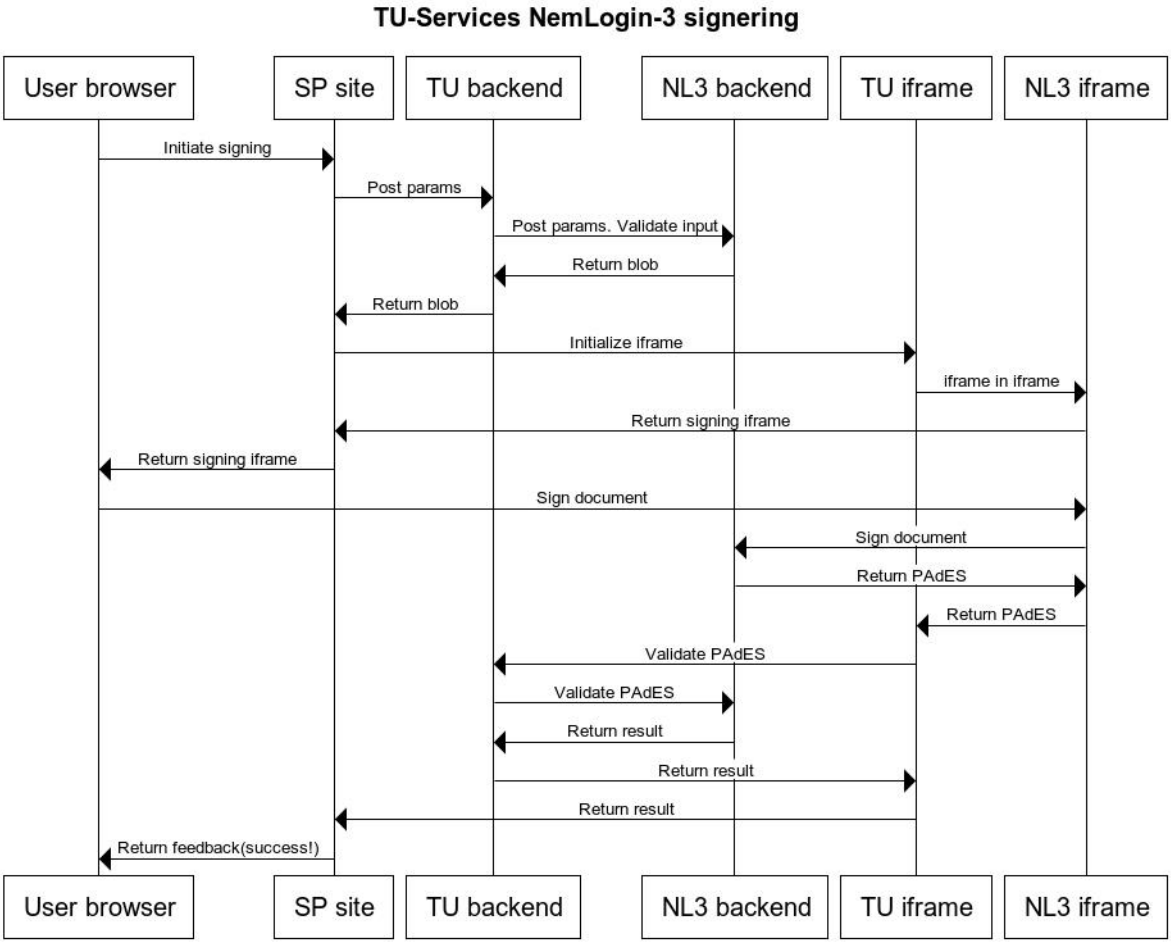

Figure 1. TU-Services NemLogin-3 signering

5.2 Signing Parameters

The standalone signing flow supports the following parameters:

Parameter	Value	Description
Title **< Required >**	{string}	A string representing the document. Typically the document title.
SsnPersistenceLevel < Optional >	Global / Session	Used to specify the persistence level of the uuid returned with the certificate from NemLog-in3. **Global**: Returns the CPR-UUID. This is the **default** option in the TU-services platform. **Session**: Returns a unique UUID. NemLog-in3 EIA lookup services are required to get information on the user with this option.
SignedDocumentFormat < Optional >	PAdES / XAdES	Used to specify the returned format of the signed document. **PAdES**: PDF format. This is the **default** option **XAdES**: XML format *(will be supported in a later version)*

5.3 Signing result

The result of the standalone NemLog-in3 flow returns the following upon completion. These are the NemLog-in3 signing specific parameters:

Parameter	Description
Result	The PAdES directly returned from NemLog-in3.
NI3Uuid	The UUID returned in the certificate used to sign
Format	XAdES / PAdES format, of the result.
DocumentName	Self-explanatory

5.4 Error handling

5.4.1 .NET

In case of errors, a SPS.Client.Api.Exceptions.ValidationException is thrown. The exception has the following structure:

```
public string ErrorMessage { get; set; }
public string UserMessage { get; set; }
public string FlowErrorCode { get; set; }
public FlowStatus Status { get; set; }
```

ErrorMessage contains a message suitable for logs.

FlowErrorCode can contain error codes transferred from NemLog-in3.

UserMessage will be present if any meaningful message can be displayed from SPS. If empty, we recommend displaying a general error site for the end user.

For NemLog-in3 error codes, refer to the documentation presented by NemLog-in3. The documentation can be found here: https://tu.nemlog-in.dk/oprettelse-og-administration-af-tjenester/signering-kvalificeret/tekniske-egenskaber/dokumentation/

if SPS itself fails with an error. Exceptions of type: SPS.Client.Api.Exception.BackendException is thrown. This is just a subclass of System.Exception. The message in the exception will contain information on the error.

5.4.2 JAVA

In Java, an exception of type: dk.signaturgruppen.sps.frame.ValidationException is always thrown on errors. This has the same structure as the one mentioned in.NET

Typically, an error from NL3 or MitID will have a FlowErrorCode and an error message corresponding to that error code.

In case SPS fails, the ValidationException will contain either a sufficiently descriptive message, or a stack-trace.

6 Contact information

Info	
Mail	support@signaturgruppen.dk
Phone	+45 70256425
Website	https://www.signaturgruppen.dk

7 Version Info

Version	**Update**
Version 1	Document created
Version 2	Added NemLog-in3 support
Version 2.1	Added signing parameters section
Version 2.2	Added some error handling and java details
Version 3	Added MitID acceptance flow