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 NemID and NemLog-in3 signing infrastructures.

This is done through either the Document API (described in section 3) for both the NemID 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 NemID PID/RID numbers.

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.

The Document API also allows for using both NemID 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 NemID and NemLog-in3 signature formats, infrastructure and challenges.

1.1 NemID signatures

When signing a document (PDF, text, xml or html) with NemID, a XMLDSig is created wrapping the original document. This has multiple implications for signatureflows.

A XMLDSig is a digital signature in XML format, wrapping the original document, user certificate and other signed properties into a single new document. This document cannot be edited without breaking the signature.

If you need multiple signers on the same document you then have to build a system that manages multiple parallel NemID signatures on the exact same document and creates a datamodel with a NemID XMLDSig on the same original document for each signer. These signatures are all standalone signatures on the original document, but together represent the desired effect of having multiple signers on the same document.

This is the primary use of the TU Services Document API, which let systems create and manage signing flows within the NemID infrastructure with multiple signers.

In addition to letting systems extract the set of signatures and signers for a specific document, the Document API supports the construction of a PDF Advanced Electronic Signature (PAdES), which is a signed PDF document, wrapping the entire signature flow with all NemID signatures and verifiable by anyone using Adobe Reader.

1.2 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 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 in the same way, as it is done using NemID. 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.

2 TU-services Setup

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

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)

3 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 our demo application found at https://authentication.signaturgruppen.dk for a more detailed example of the Document 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.

3.1 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();
```

3.2 Sign Document

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

1: Create the flowbuilder

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

Alternatively, a flowbuilder using NemLog-in3 can be created.

```
var flowBuilder = new NL3DocumentFrameBuilder(docGuid).SignerId(username1);
var scriptBlock = flowBuilder.GetScript("pathToYourValidationHandler");
var frameUrl = 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 NemID/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 docAuthInfo = new NemIDFlowValidator().Validate(result);
```

Alternatively, using NemLog-in3:

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

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

3.3 Signing parameters

The signing parameters are used to control parts of the NemLogin document signing. The current possible parameters are:

Parameter	Value	Description
ssnPersistenceLevel	Global / Session	Used to specify the persistence level of the uuid returned with the certificate f **Global**: Returns the CPR-UUID. This is the *default* option in the TU-services **Session**: Returns a unique UUID. NemLog-in3 EIA lookup services are requi

Signing parameters are configured in the following way:

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

3.4 Extended validation of signer properties

If additional validation is needed of signer properties like danish CPR, required NemID PID or RID, this can be specified on the NemIDDocumentFrameBuilder as follows

```
var flowBuilder = new NemIDDocumentFrameBuilder(docGuid)
   .SignerId(username1)
   .SignerPid("PID") <- optional
   .SignerRid("RID") <- optional
   .SignerCpr("CPR"); <- optional
```

If specified, the required CPR, PID or RID is verified against the constructed NemID signature and if a mismatch is found an error is thrown.

In much the same way. 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.

3.5 Get document summary

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

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

3.6 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 NemID signatures and info of signers and the document. This object contains all the actual digital signatures (NemID XMLDSig's) as well as the original PDF bytes.

3.7 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 NemID signatures.

3.8 Get all

To retrive a summary-list of all active documents

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

3.9 Get pending

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

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

3.10 Get signed

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

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

3.11 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);
```

3.12 Automatic cleanup

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

4 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.

4.1 Standalone signing

1: A simple NemLog-in3 signature flow can be initiated with the following line.

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

or with simple text

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

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 NemID/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. Refer to the SPS.Client documentation for error handling and logging.

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

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

4.2 Signing Parameters

The standalone signing flow supports the same parameters specified in section 3.3.

5 Contact information

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

6 Version Info

Version	**Update**
Version 1	Document created
Version 2	Added NemLog-in3 support
Version 2.1	Added signing parameters section