1 Introduction

This document is the technical reference for Signaturgruppens SPS.Client library.

The SPS.Client consist of a single.Net DLL (SPS.CLient.dll), henceforth referred to as the client library.

The API and configuration for the client library is described in this document.

2 Requirements

The current version of SPS.Client requires.Net 4.5.2 and the JSON.net 8.0.0.0 library (http://www.newtonsoft.com/json), which is bundled with the client.

In your solution, reference all dlls in the client library release dll (SPS.Client.dll, SPS.Core.dll and Newtonsoft.Json.dll).

It might be required, that you set the following in your web.config, if you are using different versions of NewstonSoft.

```
<runtime>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
  <dependentAssembly>
   <assemblyIdentity name="Newtonsoft.Json" culture="neutral" publicKeyToken="30ad4fe6b2a6aeed" />
   <bindingRedirect oldVersion="0.0.0.0-8.0.0.0" newVersion="8.0.0.0" />
  </dependentAssembly>
 </assemblyBinding>
</runtime>
```

3 Configuration

The configuration for the client is configured either in the web.config file or dynamically in code.

3.1 Web.config configuration section

The client library expects the following configuration section in the web.config file:

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

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

3.1.1 Configuration settings

The client section expects the "CredentialsJsonFile" attribute and allows for NemID specific settings in the NemID child-section.

The CredentialsJsonFile attribute points to a JSON file, which is created either by the HOSTED backend administration site or by the accompanying UI-tool - if using a local backend. The CredentialsJsonFile contains the certificates and paths needed to make a secure connection to a specific backend. The settings in these files, should not be changed.

If the SPS.Client is used to integrate to Signaturgruppens hosted backend, the CredentialsJsonFile is generated by your administrator with access to the Signaturgruppen administration site.

If the SPS.Client is used to integrate to a local SPS.Backend application, consult the SPS.Backend documentation for the creation of the CredentialsJsonFile.

3.1.2 NemID configuration

Specying the optional NemID child-section to the SPS.Client section, allows for some NemID specific configuration settings:

NemID	
RememberUserId	True/False (DEFAULT:True)

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

3.1.2.1 RememberUserId

If enabled, enables the user to select "remember my username" in the NemID client. It is implemented using cookies, which are set automatically by the client library.

3.2 Dynamic configuration

The client has support for setting the configuration in code either per application or per session.

The client loads the configuration in the following order: session ?? static ?? web.config.

The available methods are availble from the static class SPS.Client.Configuration.ConfigSetter.

```
ConfigSetter.SetStatic(SpsConfiguration configuration);
ConfigSetter.SetInSession(SpsConfiguration configuration);
```

The SpsConfiguration class is constructed using the same JSON files as a string argument.

3.2.1 ConfigSetter.SetStatic(SpsConfiguration configuration)

The static configuration allows application wide configuration in code and is equivalent to configuring it via the web.config file.

3.2.2 ConfigSetter.SetInSession(SpsConfiguration configuration)

The session configuration allows configuring the client on a per session basis. This enables different configuration for different users.

4 Logging

No external logging framework is used in the client library but the client supports logging by providing a set of logging actions which enables the registration of existing log frameworks to be used by the client library.

Following is an example of how to register standard logging methods using Log4Net in the Application _ Start in Global.asax:

```
//from Global.asax:

protected void Application_Start(object sender, EventArgs e)
{
        var logger = LogManager.GetLogger("Signaturgruppen.SpsClientLogger");
        SpsClientLogger.DebugAction = logger.Debug;
        SpsClientLogger.InfoAction = logger.Info;
}
```

This enables loggin without any additional dependencies and works with any logging framework and is an optional feature.

The backend will log all required information regarding login, signing, validation, revocation checks etc.

A full audit log are generated at the backend which supports the extended reporting requirements from DanID when using file or hardware based NemID flows

The backend is also able to log the XMLDSig (XML-signatures) generated by the NemID flows. These are also avaiable through the API.

5 HTML 5 Web Messaging and iframes

The flow works by setting up an iframe and then communicating with the backend using the HTML 5 Web Messaging JavaScript API through an iframe.

The client library generates the code needed to enable communication with the backend through an iframe. As an example, the following code snippet creates the signed parameters for the backend as well as the JavaScript needed for the communication

```
var frameScript = new NemIDFrameBuilder().GetScript("UrlToResultHandlePage");
```

We refer to the demo for details on how this is used.

The browser requirements for this setup are the same as for the NemID JavaScript client, thus if the users browser supports the NemID client, it supports the client library.

6 Test setup

Signaturgruppen has hosted a backend available when testing on https://tuservices-test.signaturgruppen.dk/backend

The demo application available at https://authentication.signaturgruppen.dk is configured to this test backend when you start it the first time.

It is possible to use the demo setup to get NemID up and running in your application(s) and test how NemID and our software works before any agreements with us or DanID has been made.

The client library can be used to integrate to Signaturgruppens hosted backend and to Signaturgruppens local backend (if you host it yourself).

7 Client flows and services

We support all NemID login and signing flows, which include the NemID JavaScript client using OTP cards (both Standard Mode and Limited Mode) and the OpenSign applet for key and hardware based login and signing flows.

7.1 CPR Services

The client library supports the CPR services available through the NemID infrastructure. If you are a private service provider you are restricted to the CPR Match service, which lets you validate a given CPR paired with the Personal ID (PID) from a successfull NemID authentication flow.

7.1.1 CPR (non-private service providers)

If you have a CPR agreement with DanID it is a simple configuration setting at the backend which enables the CPR service and sets the CPR in the response from the backend and makes it available through the NemIDValidationResult (see below).

The CPR is encrypted and decrypted transparently using RSA-SHA256.

7.1.2 CPR Match (private service providers)

The CPR Match service is available at the backend as a RESTful service available transparently through the client library. A CPR Match agreement with DanID is required.

There are two ways to call the CPR Match service using the client library.

The NemIDFlowResult.AuthenticationInfo object contains the method:PidCprMatch(string cpr) (in namespace SPS.Client.Api.Extensions), which returns true or false. (See below for more details on the result object). When a response is received from the backend, the PID from the user certificate is already part of the response, and thus only the CPR number is needed in order to verify if the PID matched the given CPR.

Note, that you need to include the following using statement in order to use the extensionmethod:

```
using SPS.Client.Api.Extensions;
```

The CPR Match service is also avaible as a "pure" backend-to-backend service using the following

```
var service = new PidCprMatchService();
```

```
var cprMatchesPid = service.MatchPidAndCpr(string pid, string cpr);
```

The MatchPidAndCpr(string pid, string cpr) method returns true or false. All communication is secured by https and signed and encrypted using RSA-SHA256.

Or a variant returning a more detailed response

```
var service = new PidCprMatchService();
var matchResult = service.MatchPidAndCprResponse(pid, cpr);

matchResult:
{
  Match : bool
  StatusCode : string
  StatusCodeDescription : string (english)
}
```

7.2 Sign _ Properties

If you need to add additional sign properties to the resulting XMLDSig document, you can do so by adding the Sign _ Properties property to the parametergenerator, as demonstrated here:

```
var signProperties = new Dictionary<string, string> {{"key", "value"}, {"keytwo", "valuetwo"}};
new NemIDFrameBuilder().SignProperties(signProperties):
```

7.3 NemID PDF Service

The client library provides an API for verifying if a PDF is valid for NemID signing. Use the following:

```
var verifyPdfCheck = new NemIDPDFService().VerifyPdf(pdfBytes);
```

The response contains the following structure:

```
public class NemIDPdfVerificationCheck
{
   public bool IsValid { get; set; }
   public string ErrorMsg { get; set; }
}
```

8 Backend response

The response received from the backend through the iframe is validated using the client library, resulting in an object consisting of the validation result and all the information needed from the specific client flow.

Following snippet of code creates a standard NemID validator which parses and validates the response string received from the backend through the HTML5 WebMessaging API:

```
var nemidValidationResult = new NemIDFlowValidator().Validate(result);
```

The NemIDFlowValidator.Validate() method throws an exception if the result is not as expected or if the result

does not successfully validate as a valid and expected NemID response. If a result object is returned, the user can be logged in or the document has been signed successfully.

8.1 Backend response error

In case of errors, an SPS.Client.Api.Exceptions.ValidationException is thrown, with the following properties:

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

The NemIDValidationResult object contains the validation result and all the data you need from the NemID flow.

An example is a NemID login flow, where the result object contains validation status and if successfull all the certificate info from the NemID user including: PID/RID and Common Name.

See the demo for all relevant examples, including demonstration of how to use the CPR-MATCH service.

8.1.1 ErrorMessage

Contains an errormessage sutable for your log.

8.1.2 UserMessage

Contains a user friendly message which can be safely displayed as html-text. No dynamic html content will be returned and in case of links, the link will always be a target=" _ blank" link.

If the UserMessage is empty, we recommend a generic error page for your users. We will try to have all errors where meaningful error messages can be displayed for the user covered.

9 Flow overview

To illustrate how the different parts work together, we provide a short overview of the overall flow here.

All communication are done via https and are thus encrypted.

1. The client website opens an iframe to the backend and sends the generated and signed parameters to the backend utilizing the HTML 5 Web Messaging JavaScript API.
2. The backend validates the parameters received.
3. The requested flow/client is started and generated parameters used. The user is interacting directly with the service inside the iframe using a secure connection.
4. As an example: The NemID client is started inside the iframe which transparently lets the user interact with the NemID client, as-if the client was hosted directly on the client website.
5. When the user has entered his credentials the response is validated and all required logs created at the backend.
6. A response is created and sent to the client website through the iframe using the HTML 5 web massaging API.
7. The response is validated using the client library, after which all required information is available through the client library API.
8. All messages are signed and sensitive information is encrypted. Our library transparently validates and de-/encrypts the messages.

10 Backend variants

The client library integrates to Signaturgruppens SPS.Backend application, which is available in two flavours.

10.1 Hosted backend

Signaturgruppen has a full-featured hosted variant of the SPS.Backend application available using the SPS.Client library. With this setup only the SPS.Client is needed in order to implement all NemID variants on your site.

Additional API's and services are available through the SPS.Client when using the hosted backend - contact us for more information.

In this setup all the responsibility of updating, maintaining and hosting the backend is handled by us and in most cases, updates will be applied without any required action from the integrating client.

10.2 Local backend

A local backend - or standalone backend - is available, which lets you run the SPS.Backend application in your own environment.

It is the exactly same client library which also lets you switch from one setup to the other with minimal changes.

11 Contact information

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