Welcome to Smallpdf

Ready to take document management to the next level?

1 Digital Documents—All In One Place

With the new Smallpdf experience, you can freely upload, organize, and share digital documents. When you enable the 'Storage' option, we'll also store all processed files here.

2 Enhance Documents in One Click

When you right-click on a file, we'll present you with an array of options to convert, compress, or modify it.

3 Access Files Anytime, Anywhere

You can access files stored on Smallpdf from your computer, phone, or tablet. We'll also sync files from the Smallpdf Mobile App to our online portal

4 Collaborate With Others

Forget mundane administrative tasks. With Smallpdf, you can request e-signatures, send large files, or even enable the Smallpdf G Suite App for your entire organization.

Sign here

pdfy@small.com

Implementation Guidelines for NemID

(OCES)

Table of Contents

Version History

31 January 2014	Version 1.3	OYVMO
25 March 2014	Version 1.4	PHJER
14 April 2014	Version 1.5	OYVMO
12 May 2014	Version 1.6	OYVMO
19 May 2014	Version 1.7	RPLAU
20 May 2014	Version 1.8	OYVMO
22 May 2014	Version 1.9	PHJER
3 June 2014	Version 2.0	KSANO
6 June 2014	Version 2.1	PHJER
1 August 2014	Version 2.2	OYVMO
19 August 2014	Version 2.3	OYVMO
3 September 2014	Version 2.4	KSANO
9 September 2014	Version 2.5	OYVMO
20 October 2014	Version 2.6	OYVMO
28 October 2014	Version 2.7	OYVMO
30 October 2014	Version 2.8	OYVMO
29 January 2015	Version 2.9	KMAIB
9 February 2015	Version 3.0	STNOR
2 March 2015	Version 3.1	PKAJB
24 June 2015	Version 3.2	PCKOC
August 10 2015	Version 3.3	RPLAU
August 25 2015	Version 3.4	PKAJB
September 10 2015	Version 3.5	RMELD
September 16 2015	Version 3.6	KPERS
September 30 2015	Version 3.7	PKAJB
February 5 2016	Version 3.8	MDCHR

March 30 2016	Version 3.9	PCKOC
April 04 2016	Version 4.0	ABHAN
April 05 2016	Version 4.1	ABHAN
April 07 2016	Version 4.2	ABHAN
April 13 2016	Version 4.3	ABHAN
April 15 2016	Version 4.4	ABHAN
April 19 2016	Version 4.5	PCKOC
April 25 2016	Version 4.6	PCKOC
May 18 2016	Version 4.7	PKAJB
June 18 2016	Version 4.8	KMAIB
September 14 2016	Version 4.9	MMELI
February 9, 2017	Version 5.0	MMELI
October 30 2017	Version 5.1	RPLAU
January 29, 2018	Version 5.2	RMELG
April 16, 2018	Version 5.3	RKAUR
May 1, 2018	Version 5.4	RMELG
May 14, 2018	Version 5.5	PRANN

1   The Purpose and Target Group of the Document

This document is part of the NemID Service Provider Package.

The purpose of this document is to serve as the technical documentation for integrating the NemID JavaScript clients.

The document is aimed at developers and architects.

2 Introduction

This document serves as the technical documentation for integrating with the NemID JavaScript client and with the NemID CodeFile client.

Its intended audiences are developers and architects.

> Please note that the Danish Agency for Digitisation has developed a special Service Provider package, called "LSS for NemID TU-package" that provides support for tablets and smartphones for users with NemID employee signature as code file in companies using a local signature server (LSS).
>
> This feature is now fully integrated in the NemID CodeFile client and does not need a separate implementation by the individual Service Provider.
>
> The documentation of LSS for NemID is available at https://www.lss-for-nemid.dk.

2.1 Document Conventions

`Code examples and XML snippets are written using a fixed width font.`
References are included by adding the reference key after a relevant sentence, e.g. [XMLDsig]. The list of references can be found in appendix A.

Most URLs for accessing the NemID JavaScript client and other NemID systems are documented in the document "Configuration and setup" which is available as part of the Service Provider documentation [SP-docs].

2.2 Integrating the NemID JavaScript client overview

This section is meant as a way for Service Providers to gain a quick overview of the development effort required to integrate with the NemID JavaScript client.

2.2.1　　Iframe integration

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

**Figure 2-1 - The NemID JavaScript client in limited mode 320*460
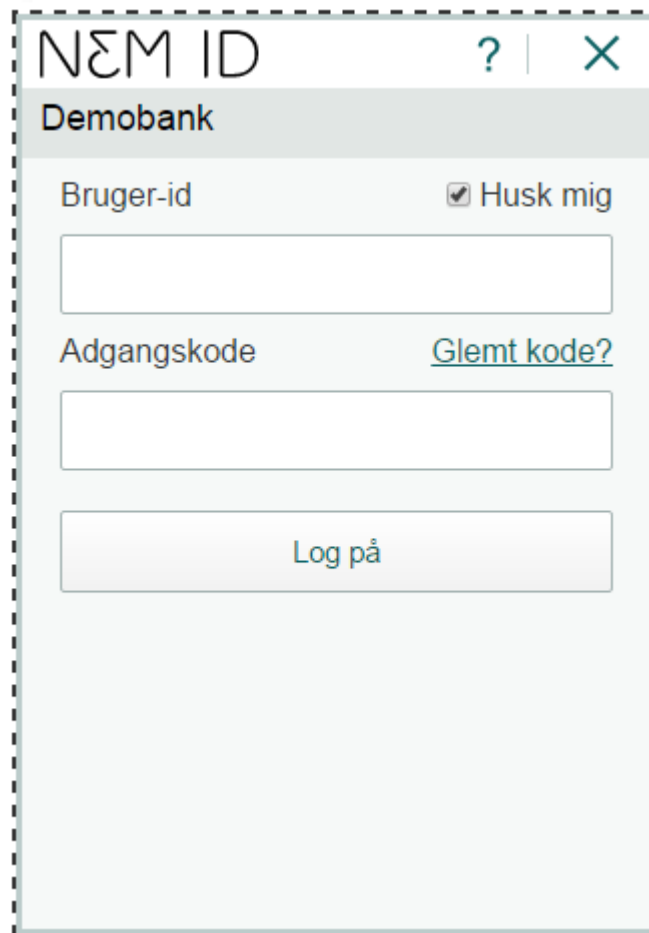with the recommended option of "Remember userid". The dashed
line indicates the extent of the iframe.**

An `<iframe>` element does not allow its content to expand beyond its
borders, which necessitates that an area sufficient for every possible
screen size is allocated when it is created.

Please see Figure 3-2 for details about minimum width and height of the
iframe.

Note that the NemID JavaScript client does not support "quirks mode" for Internet Explorer. Unfortunately, if the containing page is rendered in quirks mode, then all `<iframe>` elements in that page are also rendered in quirks mode. Therefore, the Service Provider's page must avoid quirks mode. Take particular care regarding the HTML DOCTYPE declaration, and consider adding a meta tag to the HTML header like:

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

Example `<iframe>` element for integrating the NemID JavaScript client with the Service Provider's page:

```
<iframe id="nemid_iframe" title="NemID" allowfullscreen="true"
scrolling="no" frameborder="0"
style="width:320px;height:460px;border:0" src="https://…"></iframe>
```

Screen readers (accessibility tools) depend on the title attribute for describing iframes, and we suggest that it is set simply to "NemID".

Note also that the `allowfullscreen="true"` attribute should be set on the iframe element whenever the JavaScript client will initiate PDF signing flows, so that full screen viewing of the sign text will be available to the user.

Chapter 3. NemID JavaScript client integration describes how to integrate the NemID JavaScript client in more detail.

2.2.2　　Iframe size

As stated above the size of the iframe that the Service Provider creates for the NemID JavaScript client is determined by the different iframe properties controlled by the Service Provider and the minimum sizes defined by Nets.
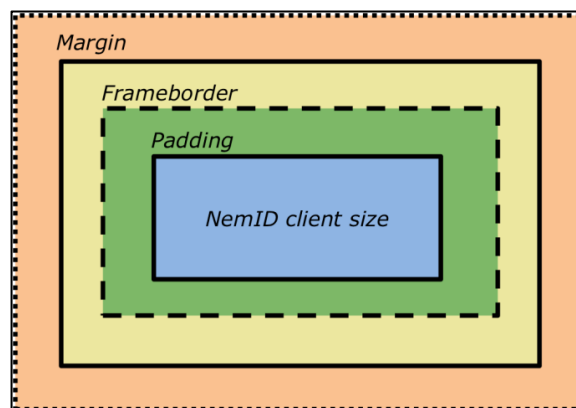
Figure 2-2

The above figure 2-2 indicates that the Service Provider must consider the sum of the iframe attributes margin, border and padding together with the minimum size indicated by the Nets, i.e. the size of the iframe must be the required client size plus the padding, border and margin.

There are requirement in regards to the size of the Blue area in the picture above. Please find more details in Figure 3-2.

Height min = client minimum height + top(border+margin+padding) + buttom (border + margin + padding)

Width min = client minimum width + left (border+margin+padding) + right(border+margin+padding)

2.2.3   Parameters

Client parameters are supplied as a JSON structure. The parameters are transmitted to the client using the `postMessage` functionality that is part HTML5 [Web Messaging].

All parameter names and most parameter values are case insensitive.

3 NemID JavaScript Client Integration

The NemID JavaScript client is integrated in the Service Provider's page using an `<iframe>` element. Communication between the NemID JavaScript client in the `<iframe>` and the Service Provider's page is carried out using the HTML5 APIs for cross-domain communication[1] [Web Messaging].

Because the NemID JavaScript client's iframe element must exist within a containing HTML document, any meta tags that are included in the head element of the HTML document might also affect the NemID JavaScript client. The Service Provider can thus control some aspects of the user experience:

- **<meta name="viewport">**
  The viewport meta tag controls the dimensions and resolutions of the browser viewport, and can affect how mobile browsers display and scale the content. A useful setting can be:
  <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">

The URL of the client, which must be used as the `<iframe>` element's SRC attribute, is specified in the document "Configuration and setup" [SP-docs].
Please note that the <random> postfix of the URL must be a constantly changing number such as system time in milliseconds. Its purpose is to prevent caching of resources in the client.

The NemID JavaScript client is initialized by taking the following steps

1. The Service Provider page is opened with an iframe pointing to the NemID JavaScript client URL.

2. The NemID JavaScript client transmits a `SendParameters` message to the Service Provider page to indicate when it is ready to receive its parameters.

---

[1] http://html5test.com/Compare/feature/communication-postMessage.html

3. The Service Provider page transmits a message containing the client's initialization parameters.

4. The client initializes based on the parameters and the user can interact with it.

5. Based on the user's action, the Service Provider page receives either a signed message or a response code indicating what prevented the user from completing the operation.

The flow is illustrated in Figure 3-1 below.
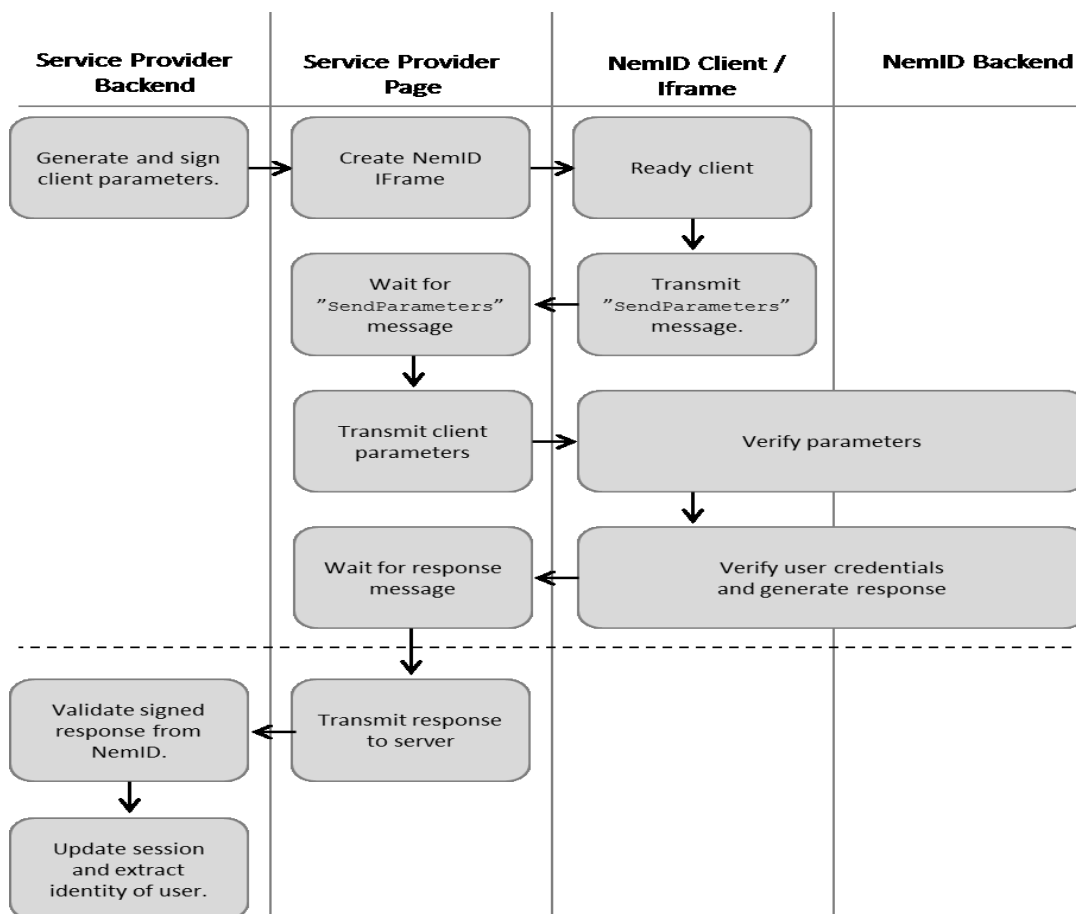
Figure 3-1 – The NemID JavaScript client integration flow

3.1 Parameters

The NemID JavaScript client receives the parameters in a JSON object. Each parameter consists of a name and a value, both of which are strings. An example of parameters in a JSON object is given below.

```
{
    "TimeStamp": "2014-02-25 08:37:48+0100",
    "ClientFlow": "ocessign2",
    "Signtext_Format": "html",
    "Params_Digest": "UO5hbuKUcfgjxdfAWzjy04G5hyJ2vUa9GdR3yWwXKm0=",
    "Digest_signature": "FHbj9p8k … CIjAmi9TblUZqJI=",
    "SP_cert": "MIIE/DCCBGWgAwIBAgIEQDgI … 358IwXJUJWnVOln+o=",
    "Signtext": "PGh0bWw+PGJv … Ym9keT48L2h0bWw+"
}
```

The ordering of the parameters in the JSON object holds no significance. All parameter names are case-insensitive. Some values, e.g. base64 encoded strings or URLs, are case-sensitive.

The following tables contain a description of the parameters.

Name	Description	Allowed Values
CLIENTFLOW *mandatory*	Determines which NemID flow to start	• OCESLOGIN2 2 factor OCES login • OCESSIGN2 2 factor OCES signing
CLIENTMODE	Previously determined the client mode, either standard or limited, but is now ignored. Use the launcher url instead to control client mode.	• STANDARD • LIMITED
CREDENTIAL_UPDATE	Used to indicate that the user wants to change credentials (user alias and / or password)	• PASSWORD => User is prompted to change password • ALIAS => User is prompted to change user alias and password • Any other value => User is not prompted to change any credentials (default)
DIGEST_SIGNATURE *Mandatory*	Base64 encoded RSA signature of the calculated parameter digest. See chapter 3.2 for information on this parameter.	Signature on the calculated digest

DO_NOT_SHOW_CANCEL	Used to prevent the Cancel button from being displayed in the UserID Password screen The paramter is only relevant for standard mode, the cancel X is always shown in responsive mode	• TRUE => The cancel button is not displayed • Any other value (default) - The cancel button is displayed.
ENABLE_AWAITING_APP_APPROVAL_EVENT	Indicates if the JS client should send out an awaiting app approval event. The event can be caught outside the JS client using the event handler set-up from section 3.3 where command will be "AwaitingAppApproval".	• TRUE => Enable awaiting app approval event • Any other value or unset (default) => No event

LANGUAGE	Client language	• DA
		  Danish - default
		• EN
		  English
		• KL
		  Greenlandic

ORIGIN *Optional, but highly recommended*	The origin of the Service Provider site which will send parameters to the NemID JavaScript client. The NemID JavaScript client will abort with APP001 or APP007 if a postMessage command is received from any other origin.	URL describing the domain, in the format: protocol://ip:port Example: The ORIGIN for an Service Provider login-site at https://example.com/nemidlogin.aspx would be https://example.com
PARAMS_DIGEST *Mandatory*	Base64 encoded representation of the calculated parameter digest. See chapter 3.2.for information on this parameter.	Calculated digest of parameters

REMEMBER_USERID *Optional, but highly recommended*	Base64 encoded token returned from the client when the user chooses to remember his user id. At next login/signing this parameter must be specified in order to enable the remember user id functionality.	Prior returned token or the empty string if the user has not remembered his user id previously.
REMEMBER_USERID_INITIAL_STATUS *Optional, but highly recommended*	Indicates that the "Remember userid checkbox" should not be initially checked. This is only relevant in responsive mode and when REMEMBER_USERID is also set.	• FALSE => "Remember userid checkbox" is **not** checked. • Any other value or unset (default) => "Remember userid checkbox" **is** checked
SIGNTEXT *It is mandatory to specify either SIGNTEXT or SIGNTEXT_URI for all signing flows*	Base64 encoded representation of the actual text signed by the user. Must be in the format specified by the SIGNTEXT_FORMAT parameter.	
SIGNTEXT_FORMAT *Optional (relevant for signing flows only)*	The format of input given by SIGNTEXT parameter	• TEXT plain, unformatted text (default value) • HTML • XML • PDF
SIGNTEXT_MONOSPACEFONT	Indicates that plaintext should be rendered with a mono-spaced font (to allow for indentionbased formatting)	• TRUE => Mono space font is used for plaintext • Any other value => Default font is used
SIGNTEXT_REMOTE_HASH *Mandatory if SIGNTEXT_URI is specified*	Base64 encoded SHA256 hash of the remote PDF document	

SIGNTEXT_TRANSFOR MATION *Mandatory if SIGNTEXT_FORMAT is XML*	Base64 encoded XSLT stylesheet that transforms the XML signtext into a HTML document, which is displayed to the user for signing	
SIGNTEXT_TRANSFOR MATION_ID	Identifier for XML stylesheet	
SIGNTEXT_URI *It is mandatory to specify either SIGNTEXT or SIGNTEXT_URI for all signing flows*	URI used to load a PDF document asynchronously	URI must be well formed. Only http or https are supported
SIGN_PROPERTIES	XMLDSIG properties to be included in signed response	
SP_CERT *Mandatory*	Base64 encoded DER representation of the certificate used for identifying the OCES Service Provider	Certificate must be issued by a Nets-DanID trusted Certificate Authority
SUPPRESS_PUSH_TO_ DEVICE *Optional*	Indicates that a notification should NOT be pushed to a mobile device. The device must pull the notification.	• TRUE => mobile device must pull the notification • Any other value or unset (default) => push notification to mobile devices
TIMESTAMP *Mandatory*	Current time when generating parameters. The timestamp parameter is converted to UTC and must match the NemID server time. NemID accepts timestamps within the boundaries of ±3 minutes.	Current time expressed as one of: • yyyy-MM-dd HH:mm:ssZ E.g. 2014-06-03 07:09:13+0200 or 2014-06-03 09:09:13CEST. The timezone can be left out, in which case server time will be assumed. • Epoch Milliseconds since 1970-01-01 00:00:00 Both plaintext and base64 encoded is acceptable.

TRANSACTION_CONTEXT **Optional**	Base64 encoded text describing the transaction. Will be part of the notification sent to a mobile device and will be displayed in the NemID code app when the user approved the login/signing.	Only used if the user approved the transaction in the code app. The string can be max 100 chars long and if not provided a standard message will be displayed in the code app e.g. Logon: "Handling hos ##SPNAME##" Signing: "Handling hos ##SPNAME##"

3.2 Parameter integrity

To ensure the integrity of the parameters in transit between the Service Provider and the NemID JavaScript client, they must be signed by the Service Provider.

The process for securing the client parameters is

1. The Service Provider collects the list of client parameters. The list is normalized (see section 3.2.1) into a string, and the SHA-256 digest value of the string's UTF-8 representation is calculated.

2. The digest value is signed by the Service Provider.

3. The BASE64-encoded value of the digest and the signature are added as the parameters PARAMS_DIGEST and DIGEST_SIGNATURE.

4. All parameters are collected in a JSON-message and sent to the NemID JavaScript client.

5. The client reads the parameters and normalizes them, excluding the digest value and the signature parameters. The digest value is verified.

6. The client sends the digest and the digest signature to the NemID server, which verifies the signature using the public key of the Service Provider. The certificate must be a VOCES certificate that has been issued as part of the NemID enrollment process.

3.2.1   Parameter normalization

The digest of the client parameters are calculated from a normalized version of the parameters.

The process for normalizing the parameters is

1. The parameters are sorted alphabetically by name. The sorting is case-insensitive.

2. Each parameter is concatenated to the result string as an alternating sequence of name and value:

   $name_1 \; || \; value_1 \; || \; name_2 \; || \; value_2 \; || \; \dots \; || \; name_n \; || \; value_n$

The result of the concatenation is the normalized list of parameters.

Below is a normalized version of the parameter example that was given above:

```
ClientFlowocessign2SigntextPGh0bWw+PGJv...Ym9keT48L2h0bWw+Signtext_F
ormathtmlSP_certMIIE/DCCBGWgAwIBAgIEQDgI...358IwXJUJWnVOln+o=TimeSta
mp2014-02-25 08:37:48+0100
```

3.3 Start-up and handling responses

Figure 3-1 contains an overview of the communication flow between a Service Provider and the NemID JavaScript client.

The nature of the HTML5 `postMessage` API makes it necessary to use a synchronization message to signal to the Serivce Provider page when the client is ready. This is what is referred to as the `SendParameters` message above. The Service Provider must wait until it receives that message, and then transmit the parameters to the client.

The following JavaScript, intended to run in a Service Provider's page, is a simple example of how to handle the message exchange. Note that the value of `nemid_server_url_prefix` should match the iframe src attribute for the NemID JavaScript client as specified in the the Configuration and Setup document [SP-docs].

```
function onNemIDMessage(e) {
    var event = e || event;

    var win =
        document.getElementById("nemid_iframe").contentWindow,
        postMessage = {}, message;

    message = JSON.parse(event.data);

    if (event.origin !== nemid_server_url_prefix) {
        window.alert("Received message from unexpected origin : " +
event.origin);
        return;
    }

    if (message.command === "SendParameters") {
        var htmlParameters =
document.getElementById("nemid_parameters").innerHTML;

        postMessage.command = "parameters";
        postMessage.content = htmlParameters;

        win.postMessage(JSON.stringify(postMessage),
nemid_server_url_prefix);
    }

    if (message.command === "changeResponseAndSubmit") {
        document.postBackForm.response.value = message.content;
        document.postBackForm.submit();
    }
}

if (window.addEventListener) {
    window.addEventListener("message", onNemIDMessage);
} else if (window.attachEvent) {
    window.attachEvent("onmessage", onNemIDMessage);
}
```

The code attaches an event listener to the `message` event, which is fired whenever the document receives a message from other documents [Web Messaging]. The event listener, in this case the function named `onNemIDMessage`, handles the following cases:

- The reception of a `SendParameters` message, which indicates that the client has finished its basic initialization and is ready to receive the Service Provider parameters. The parameters are taken from the `script` tag with the identifier `nemid_parameters` (not shown in the example), and sent in a message to the client using the `postMessage` function.

- The reception of the `changeResponseAndSubmit` message, which indicates that either a login or a signing flow has been completed. The code inside each handler inserts the response into a `FORM` element (not shown in the example) and submits the form.

3.4 UI Modes

The NemID JavaScript client can be started in one of two available modes: The standard mode and the limited mode. Each mode is described in the sections below.

3.4.1   Limited mode

This is the recommended mode for the NemID JavaScript client for both mobile devices and desktop clients.

Characteristics of Limited Mode and advantages compared to Standard Mode:

- Responsive GUI fitting content to the chosen iframe size

- Optimized button sizes for touch screens (Bigger buttons)

- Optimized input boxes for touch screens (Bigger input boxes)

- Help screens is positioning as an overlay the current screen (secondary screens)

- Texts in GUI has been revised in order to be more precise

- Scrolling has been minimized on small screen sizes (200*250)

- Layout has been optimized with space for the keyboard on mobile devices

- The Login procedure has been optimized, so the user needs to click fewer times.

- Signing screens optimized for mobile devices, so scrolling horizontally is avoided

- Facelifted GUI compared to standard mode

- Supports Desktop and mobile clients equally good

Limited mode was originally introduced to address the needs of Service Providers that have a page layout in which the size requirements for standard mode could not be implemented, and for embedding the client in pages designed for smaller screens, e.g. mobile devices.

A client started in "limited mode" must be given a minimum width and a minimum height, but will adapt itself to the allocated area. Minimum width and height is language depended and described in Figure 3-2. No screen in limited mode will require more space than what is initially allocated.

The client will always occupy the entire iframe size regardless of flow e.g. login, signing, first time activation and so on. As it will never require more space than it was initially allocated it can also be used in web page layouts if restrictions exits that prevent the standard mode from being used.

All flows and operations are supported in limited mode.

3.4.2   Standard mode

A client started in standard mode must be allocated a minimum width and height which is illustrated in Figure 3-2. Be aware that the client during a normal login flow will not occupy the entire iframe. However during special flows, for example first time activation, the client will expand and occupy the entire iframe. This is mentioned in Figure 3-2 as well.

All flows and operations are supported in standard mode.

It is not recommended to use the standard mode for mobile devices.

3.4.3 UI modes comparison

The following feature matrix explains the main functional differences between limited mode and standard mode:

	Standard mode (Width * Height)	Limited mode (Width * Height) Non-signing screens	Limited mode (Width * Height) Signing screens
Minimum size for Danish and English	500px * 450px	200px * 250px (*1)	320px * 460px
Minimum size for Greenlandic	500px * 450px	250px * 300px (*1)	320px * 460px
Minimum Recommended size	500px * 450px	320px * 460px	320px * 460px
Dynamic scaling of GUI to fill iframe	Signing screens only.	Yes	Yes

Figure 3-2

*1: If the width is 300px or above, then the recommended height is 460 in order to minimise scrolling within the iframe. For signing screens it is recommended to set the height, so the sign text is readable or use the maximum size of the screen on mobile devices.

3.5 NemID JavaScript client integration for Mobile Applications

For Service Providers who wish to integrate the NemID JavaScript client into a mobile application a separate mobile integration document exists (**[Mobile-doc])**. This document is accompanied by code examples for the iOS and Android platform **([Mobile-code]**). For Windows phone a short guide with implementation pointers is supplied in the mobile integration document (**[Mobile-doc])**.

3.6 Tab Index

The NemID JavaScript client uses tabindex from 51-55 in order to control the tab flow in the client.

If other UI elements are present on the same web page as the client then make sure to avoid using the tabindexes reserved by the NemID JavaScript client.

4 Authentication

An OCES authentication flow is initiated by setting the `CLIENTFLOW` parameter to `OCESLOGIN2`.

Service Providers identify themselves to NemID by including their VOCES certificate in the `SP_CERT` parameter, and by signing the parameter digest with the private key, as described in section 3.2.on parameter integrity.

NemID verifies that the certificate is valid and that the parameters were indeed signed using the corresponding private key.

A timestamp must be supplied in the TIMESTAMP parameter in order to limit the time a parameter object is valid.

Handling the response from a successful OCES authentication is identical to handling the response from a successful signing operation. The steps required to do this are covered in chapter 6.

5 Signing

This section describes the signing functionality of the NemID JavaScript client.

The following sign text formats are supported: HTML, XML, plain text and PDF.

Sign texts must be given as a base64-encoded parameter to the client. A URL to the document can also be supplied in the case of PDF documents. Upon successful signing, the base64 string is included unmodified as a property in the signed document.

If a signing operation is unsuccessful, an error code is returned to the Service Provider. The error code is given to the Service Provider base64 encoded.

The signing specific parameters are described in section 3.1.

5.1 Plain text signing

Plain text signing will display the sign text directly to the user with white space preserved. If the SIGNTEXT_MONOSPACEFONT parameter is used, a fixed width font will be used for displaying the sign text.

5.2 HTML signing

The HTML signing flow displays styled html to the user. Only a subset of HTML and CSS is allowed, and the sign text must therefore comply strictly with both HTML and CSS whitelists. Note that external CSS style sheets are not allowed. If any illegal content is found, the session is aborted with the APP002 error code.

5.2.1   Html element restrictions

The table below lists the allowed HTML elements and their attributes. No other elements, including HTML comments, can be used in a HTML sign text.

HTML Element	Supported attributes
html	xmlns
body	text bgcolor class style
head	
style	type
title	
p	align bgcolor style class
div	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

font	face size color
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

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

5.2.2 CSS Restrictions

A limited set of CSS styles are supported. Please note that external CSS files are not supported. The following CSS styles are supported:

background	background-color	bottom
color	clear	display
float	height	left
line-height	margin	margin-right
margin-top	margin-left	margin-bottom
overflow	position	right
top	width	white-space

Additionally, the following CSS families are supported, meaning that all styles matching the pattern "familyname-*" are supported:

border	font	list
padding	text	

5.3 XML signing

Signing of XML documents adds the following parameters to the client:

SIGNTEXT_TRANSFORMATION

A base64 encoded XSLT style sheet that can transform the XML document in the SIGNTEXT parameter into a HTML document that adheres to the rules outlined in section 5.2.

SIGNTEXT_TRANSFORMATION_ID

This optional parameter allows the Service Provider to include an identifier for the transformation used. The value of this parameter will be included as a property in the signed document, and may assist the Service Provider in later referencing the transformation that was used for displaying the sign text.

The parameter will be XML-encoded prior to being included in the resulting document, but will not be modified in other ways.

5.3.1 XSLT Output Method

The most predictable output from an XSLT transformation is gained by setting the output method to XML. This is the recommended setting for all XML transformations.

```
<xsl:output method="xml" omit-xml-declaration="yes" indent="no"/>
```
Read on below for an explanation.

XSLT supports three output methods, TEXT, HTML and XML.

Transformations will default to XML if nothing else is explicitly specified, unless the output document can be recognized as a HTML document, in which case the output method is set to HTML.

The XSLT specification details a number of transformations that XSLT engines must apply to output documents when the output method is HTML[2], e.g. collapse boolean attributes and adding meta elements. The level of adherence to these rules differs across browsers, and should thus be considered unpredictable.

The most predictable output from an XSLT transformation is achieved by specifying the output method to be XML, as in the following example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

---

[2] http://www.w3.org/TR/xslt#section-HTML-Output-Method

```
    <xsl:output method="xml" omit-xml-declaration="yes"
indent="no"/>
    <xsl:template match="/"><html><body><h1>Transaction</h2>
    …

</xsl:stylesheet>
```

Furthermore the resulting HTML document must start with standard top tag <HTML> otherwise it might not validate on all platforms.

```
A simple example generating valid HTML:

XML:

<?xml version="1.0" encoding="UTF-8"?>
<list>
    <item>
        <line1>DanID</line1>
        <line2>NemID</line2>
    </item>
</list>

XSL:

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="yes" indent="no" />
<xsl:template match="/">
  <html>
  <body>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Line 1</th>
        <th>Line 2</th>
      </tr>
      <tr>
        <td><xsl:value-of select="list/item/line1"/></td>
        <td><xsl:value-of select="list/item/line2"/></td>
      </tr>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

5.4 PDF Signing

PDF Signing will display the PDF document directly in the client, in the same space as used for signing other document types. Functionality is included to view the PDF in full screen, which includes navigation and scaling functionality.

	Internet Explorer 8 is not supported for PDF signing, as it does not support the necessary HTML5 features to adequately display PDF sign text. The NemID user interface will display an error message to the user if the user's browser does not support the required HTML5 features. Even so, the Service Provider should check and disallow users with incompatible browsers to start any PDF signing flow.

Only a sub set of the PDF specification is supported for signing. Refer to section 5.4.2 PDF whitelisting. It is recommended that PDF documents used for signing comply with the PDF/A standard.

The signing specific- and PDF-specific parameters are described in section 3.1.

Note in particular that:

- The parameter `CLIENTFLOW` must be `OCESSIGN2`.

- If a remote PDF document is referenced using the `SIGNTEXT_URI` parameter, there must always be a corresponding `SIGNTEXT_REMOTE_HASH` parameter containing the Base64 encoded SHA256 hash of the remote PDF document.

- In order for the PDF signing and the view document functionality to function, the `<iframe>` containing the NemID JavaScript client must have the attribute `allowfullscreen="true"`.

When PDF signing is specified, the PDF document is presented directly in the NemID JavaScript client.

5.4.1   External PDF files and Cross-origin Resource Sharing

Browsers limit the interaction between content loaded from different sites, also known as the Same Origin Policy [SOP]. This feature complicates the interaction between the NemID JavaScript client and any external PDF content provided by the Service Provider. The NemID JavaScript client will attempt to override the SOP by using Cross-origin Resource Sharing [CORS] (or the XDomainRequest object for IE8 and IE9 [XDRO]). Therefore it is necessary to add the following HTTP header to the http-response serving an external PDF intended for the NemID JavaScript client (for NemID KOPI):

```
Access-Control-Allow-Origin: https://appletk.danid.dk
```

To further provide compatibility with IE9 and XDomainRequest a PDF file must be base64 encoded. The Signtext_Remote_Hash parameter must be calculated from the PDF file before base64 encoding is applied. Other browsers not using XDomainRequest will be able to handle base64 encoded PDF as well.

Note: Do not rely on cookies being preserved in CORS requests, as this will not perform reliably across all browsers and. Any unique session-related information should be set as part of the value of the SIGNTEXT_URI parameter.

Note: In order to ensure maximum browser compatibility, all external PDF files must be served over HTTPS.

5.4.2 PDF whitelisting

For security reasons, and to ensure that a PDF document does not change in either content or appearance, only a sub set of the PDF specification is supported. In other words, not all PDF documents may be signed. A *whitelist* is used, which contain those elements from the PDF specification, which are supported and hence allowed in a PDF document.

To verify that a PDF document can be validated, a sign text validator can be downloaded from the Service Provider web site https://www.nets.eu/dk-da/kundeservice/nemid-tjenesteudbyder/NemID-tjenesteudbyderpakken/Pages/vaerktoejer.aspx. The sign text validator will validate a PDF document against the whitelist and can be installed locally for testing. The sign text validator is intended to be used on a document that is already known to be valid as a PDF document. The sign text validator validates that a document contains only supported PDF elements but does not check whether the PDF document strictly conforms to the PDF specification or check whether documents will render correctly in the user's browser. The rendering of PDF documents relies on the user's web browser for display, so it is important to test all supported browsers to verify that a PDF document will be shown correctly. To test the rendering you can use the developer web site (https://appletk.danid.dk/developers/signtextviewer.jsp) where you can test if the document will be shown correctly to the user in a particular version of a

browser. You must be registered with Nets to use the developer site (Information about becoming a NemID Service Provider is available at https://www.nets.eu/dk-da/kundeservice/nemid-tjenesteudbyder/Quick-guide-in-English).

If you intend to support code files for signing, be aware that rendering of PDF documents is handled differently in this case. See the chapter "Integration with the NemID CodeFile client" for more information.

Different tools generate PDF documents in very different ways. Adobe in one way, Microsoft in another and so forth.

Generally only elements from the Adobe PDF specification are supported. However, there are specific exceptions for the following elements used by Microsoft Office, which are also supported:

- /Workbook
- /Textbox
- /Endnote
- /Worksheet
- /Macrosheet
- /Annotation
- /Dialogsheet
- /Chartsheet
- /Diagram
- /Footnote
- /Chart
- /Slide
- /InlineShape
- /Artifact
- /Figure
- /Formula
- /Link

Fonts in the PDF document should generally be embedded in the document. However, the following fonts (or suitable substitute fonts

with the same metrics), are directly supported by all. It is therefore normally not necessary to embed these so called "base 14 fonts":

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

The following are examples of PDF elements, which are **not** supported:

OpenAction

OpenAction can be used to do something when reader shows a page. This could be jump to another page, open external documents, run external programs or creating a popup. This could potential open up for a security issue. Different viewers could also handle the command in a different way and this means that the signed PDF document is not presented in to same way for both the signee and the sender.

EmbeddedFile / FileAttachment

The viewer used to view the PDF might not handle embedded files correct and in a normal PDF embedded files are not needed. Embedded fonts are allowed as they cannot do harm to users computer.

Sound / Movie / Widget

Sounds are not used in a normal PDF, the same goes for Movies and widgets. When the PDF is presented in the signing client, the PDF is

shown as text and images only and all the interactive widgets cannot be displayed.

GoToR / GoTo E

GoToR allows a user to open another PDF and go to a specific location. GoToE allows going to a destination inside an embedded file. All the information needed for the signing should be in the PDF document.

SubmitForm / AcroForm

As the PDF document should be the same each time it is opened, forms are not allowed.

The complete PDF whitelist is described in Appendix C PDF Whitelist.

6 Response handling

6.1 The structure of the response message

At the conclusion of a successful flow, an XMLDSig message is returned to the Service Provider's page using HTML5 `postMessage` API (see section 3.3). Please refer to [XMLDsig] for the authoritative description of the standard. The typically most interesting information is briefly described below:

`<ds:KeyInfo>`	Contains the user's certificate and the complete chain of issuers' certificates.

`<ds:SignedInfo>`
    `<ds:Canonicalization Method>`
    `<ds:SignatureMethod>`
        `<ds:Reference>`
        `<ds:DigestMethod>`
        `<ds:DigestValue>`
	This is the element that is actually signed. It contains a reference to, and a digest of the `<ds:Object>` element having Id="ToBeSigned".

`<ds:Object … Id="ToBeSigned">`
    `<ds:SignatureProperties>`
        `<ds:SignatureProperty>`
	The `<ds:Object>` element is the element referenced from within `<ds:SignedInfo>`. Its `<ds:SignatureProperty>` element contains an `<openoces:Name>` and `<openoces:Value>` element pair. The remainder of this table lists the possible names.

`<openoces:Name>` RequestIssuer	The name of the Service Provider (As displayed to the user in the NemIDs GUI)

`</openoces:Name>`	

`<openoces:Name>`
`    TimeStamp`
`</openoces:Name`
	The value of the TIMESTAMP parameter from the request is returned to the Service Provider.

`<openoces:Name>`
`    action`
`</openoces:Name`
	The type of the flow that is documented by this signature ("logon" or "sign")

`<openoces:Name>`
`    signtext`
`</openoces:Name`
	Signature only, The signtext itself, that has been displayed to the user.

Please note that neither CPR nor CVR numbers are contained in the XMLDSig message. However, the PID or RID number can be found in the user's certificate, and this can be used to look up the CPR or CVR number by using the Service Provider Package (see chapter 9. NemID Service Provider package):

`SubjectDN: SERIALNUMBER=DANID:xxxxxxxxx`

6.2 Verifying the user's certificate

When the user has authenticated himself, his signature is packaged in the XMLDSig message returned to the Service Provider's page (see the previous section). The Service Provider's page would typically place the XMLDSig document in a HTML form on the website, and then submit the form to be processed by a back-end application. The XMLDSig document must be extracted from the form submission on the web server, and that is also where the signature in XMLDSig should be validated.

The Service Provider must then have the certificate validated and, where applicable, must translate the PID/RID number to a CPR number. This requires the Service Provider to run through the following steps in the case of a logon. The list will be equivalent in the case of verifying a signing:

1. Validate the signature on XMLDSig.

2. Extract the certificate from XMLDSig.

3. Validate the certificate and identify CA as OCES throughout the whole certificate chain to the root certificate.

4. Check that the certificate has not expired.

5. Check that the certificate has not been revoked.

6. Extract the PID or RID from the certificate.

7. Match or translate the PID or RID to a CPR number (translation is only available to select Service Providers).

> **Verification of the root certificate by telephone.** The Service Provider can verify whether a root certificate is correct by calling +45 72 24 70 12. By comparing the fingerprint found in the root certificate with the fingerprint read aloud on the telephone, the correctness of the root certificate can be confirmed.

The steps above will depend on whether you choose Nets DanID's security package/OOAPI for integration or whether you prefer to adapt/extend the integration yourself.
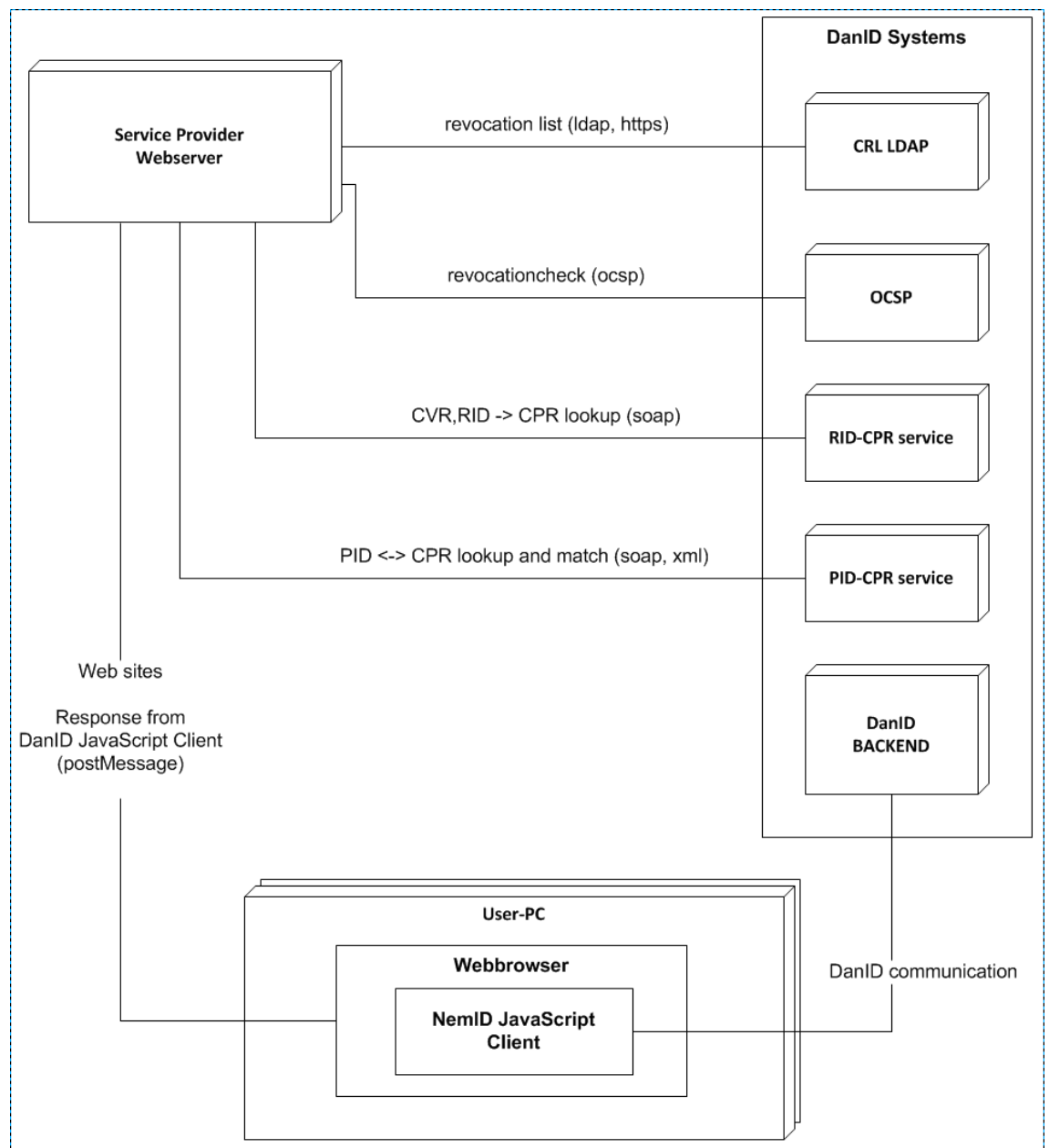
Figure 6-1 – Deployment diagram – Systems that communicate during login.

Figure 6-1 depicts systems that can communicate in a login situation. The possible protocols for the communication have been added in parenthesis. After this the Service Provider can use the OOAPI delivered by Nets DanID to perform step 1 to 7 in the before mentioned list. The revocation check can be performed in two ways. Either a revocation list (CRL) can be obtained from the CRL LDAP system, or a revocation check can be performed against the OCSP system. The RID-CPR service can be used to lookup a cpr-number given CVR-number and RID of an

employee. Accordingly, the PID-CPR service can be used to lookup a cpr-number given a PID of an employee or it can be used to match a PID against a cpr-number.

> Read more about direct integration in section 7: **Direct integration with Nets DanID's infrastructure**.

6.3 Logging

It is the responsibility of the Service Provider to store and provide the information needed to prove the correctness of a signature or an authentication at a later point of time, e.g. due to a dispute or fraud case.

The Service Provider must store the signed document from each login or signing operation. The user's certificate (which includes the PID or the RID), the time stamp etc. are included in signed documents. Service Providers should also consider storing timestamp, client IP number, user agent, and similar information.

NemID records authentication and signing operations in the user's usage log.

Note that Nets-DanID A/S does not store the signed documents.

6.4 Response codes

An error code is returned to the Service Provider, if a client operation fails to complete successfully. The error code should be used to assist the user in how to remedy the situation and accomplish what he set out to do. The list of error codes is available in the separate document NemID Error Codes, which is available as part of the Service Provider documentation [SP-docs].

7 Direct integration with Nets DanID's infrastructure

Service Providers are recommended to take Nets DanID's Service Provider package (see chapter 9. NemID Service Provider package) and OOAPI as a starting point, when choosing a method of integration with NemID. This module covers most integration options and will suffice in the vast majority of cases.

Those Service Providers who wish to develop their own interface to Nets DanID's infrastructure are referred to the following documents, which are all part of the Service Provider package:

- Introduction to NemID and the Service Provider Package.

- Specification document for the PID-CPR service.

    - A description of the PID-CPR service.

- Specification document for the RID-CPR service.

    - A description of the RID-CPR service.

- The specification document for LDAP API.

    - A description of how revocation lists can be obtained from LDAP and things you need to be aware of.

- Specification document for OCSP

    - A description of how to access the OCSP responder and what the interface is like.

8 Remember user id

The remember user-id functionality is optional and is by default not enabled. To enable it one must pass the parameter named REMEMBER_USERID to the client.

When a login or signing (1-or 2-factor) transaction is completed, a Service Provider specific token that serves as a reference to the user-id information stored server side, is returned in the XMLDSIG documents. The Service Provider is responsible for storing the token (e.g. in a cookie) and use it as input to the NemID JavaScript client in future transactions.

Valid values for the REMEMBER_USERID parameter include the empty string and a base64 encoded token returned in the XMLDSIG document as mentioned. If the user has not remembered his userid previously the empty string can be sent as parameter value. This will show the remember user id checkbox option at the userid/password screen. Please note that supplying an invalid token will have the same effect as the empty string, i.e. no userid is prefilled but the remember userid checkbox is shown at the userid/password screen.

8.1 Standard Mode

If the REMEMBER_USERID parameter is enabled and the user has chosen to let the system remember the userid then the client will help the user by only allowing digits as input. However the GUI is the same as if the user has a 4-digit password or an alphanumeric password. Please see screenshot of Login screen in Figure 8-1 below:
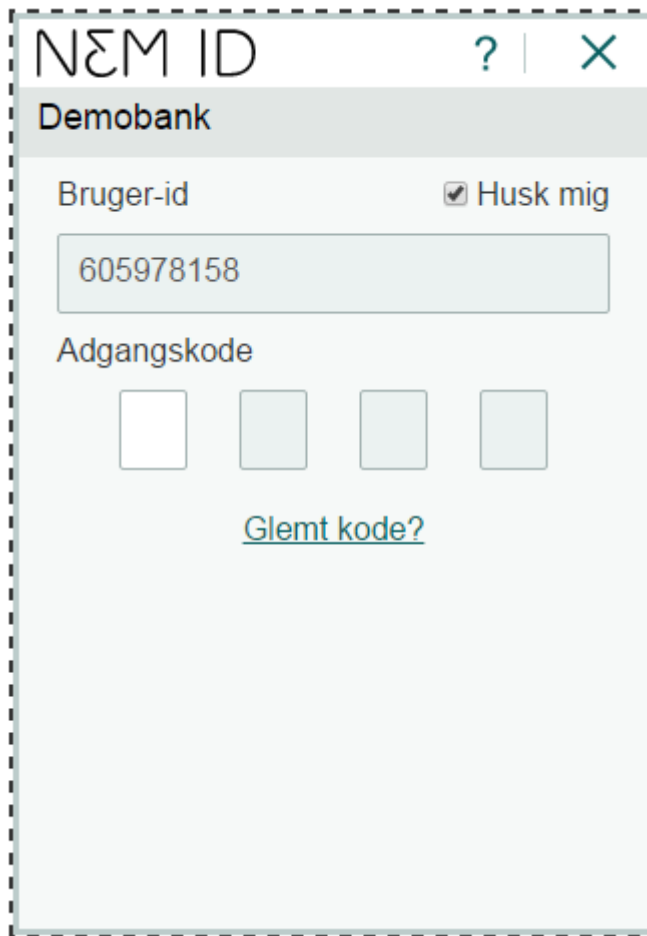
Figure 8-1 - Login screen in Standard Mode (500*450)

8.2 Limited Mode

If the REMEMBER_USERID parameter is enabled and the user has chosen to let the system remember the userid then the client will help the user by only allowing digits as input. The GUI helps the user, by showing 4 input boxes. There is no "Log på" button. This is not needed because the system will automatically submit the page when the user has inserted 4 digits. Please see screenshot of Login screen in Figure **8**-**2** below:

Figure 8-2 - Login screen in Limited Mode

9   Code app settings

Users can authenticate via the NemID Javascript client using various 2nd factors:

- Code card – A printed list of one-time codes

- Code token – An electronic device for providing one-time codes

- IVR – A voice service for providing one-time codes

- Code app – A 2nd factor confirmation app for mobile devices, providing the one-time confirmation

NemID Javascript client supports all above without any effort needed by service providers to support these. However code app integration can be enhanced by providing a textual context, see parameter TRANSACTION_CONTEXT, for presenting the context to the user in the code app.

10 NemID Service Provider package

The purpose of NemID Service Provider package is to make it easy for Service Providers to implement NemID on their websites.

10.1.1 The resources of the Service Provider package

The Service Provider package consists in addition to the documentation package of the following components:

- **ooapi-<version>-source.zip**. Source code for OOAPI.

- **ooapi-<version>.jar**. Compiled version of OOAPI.

- **ooapi-<version>-with-dependencies.jar**. Compiled version of OOAPI including the jar files upon which the OOAPI depends.

- **ooapi.net-<version>.zip**. .Net version of OOAPI.

- **ooapi.net-<version>-source.zip**. Source code for OOAPI in .Net including an example of the implementation of NemID (source version).

- **Javadoc - ooapi og sikkerhedspakke-<version>.zip**. Java doc for OOAPI and Security Package.

10.1.2 LogonHandler

The LogonHandler class offers a method by which the logon data sent by the client can be validated, while also having the PID returned for the person who is logged on (RID if it is an employee signature).

A successful validation of logon data (i.e. a call that has not given rise to any exceptions) ensures the following:

- that the signature contained in the logon data is valid.

- that the certificate contained in the logon data is valid, i.e. that the certificate has not expired or been revoked. By checking the returned status, you can see what state the certificate is in.
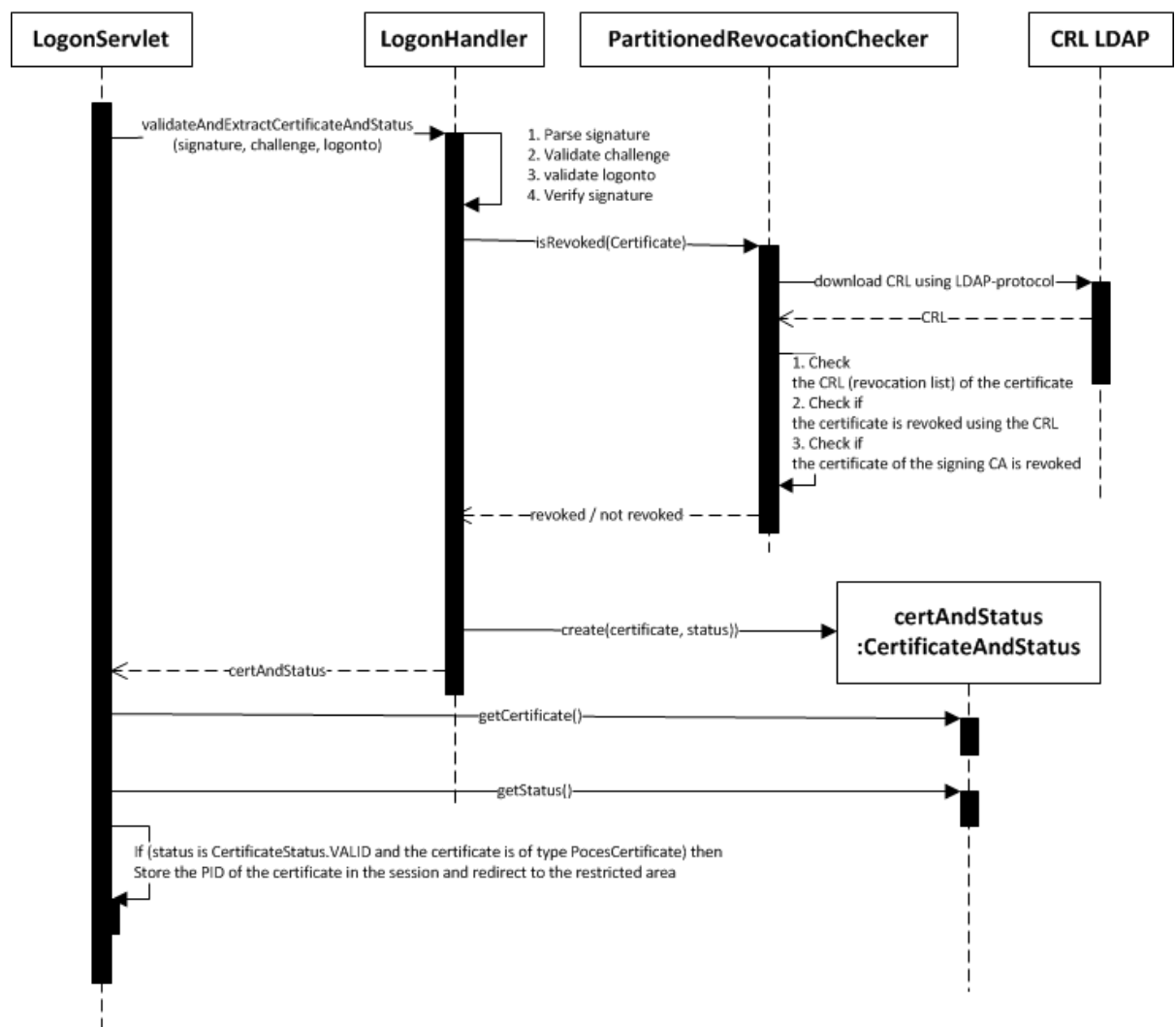
Figure 10-1 – Sequencediagram – LogonServlet handles login.

Figure 10-1 shows how the LogonServlet from Tuexample-source.zip uses the LogonHandler for validating a login. Furthermore, it shows how the LogonHandler validates the signature, challenge, and logonto parameters.

The LogonHandler parses the signature and extracts its certificate chain. The certificate chain consists of the root CA certificate, the issuing CA certificate and finally the user certificate. The certificate chain is verified by verifying that the issuing CA certificate has been used for signing the user certificate, and accordingly it is verified that the root CA certificate has been used to sign the certificate of the issuing CA. The LogonHandler then uses the PartitionedRevocationChecker class to check if the user certificate has been revoked. The

PartitionedRevocationChecker checks the user certificate to identify which partial revocation list (CRL) the user certificate belongs to. The PartitionedRevocationChecker retrieves the correct revocation list from the CRL LDAP system. If the user certificate is present in the revocation list the certificate has been revoked.
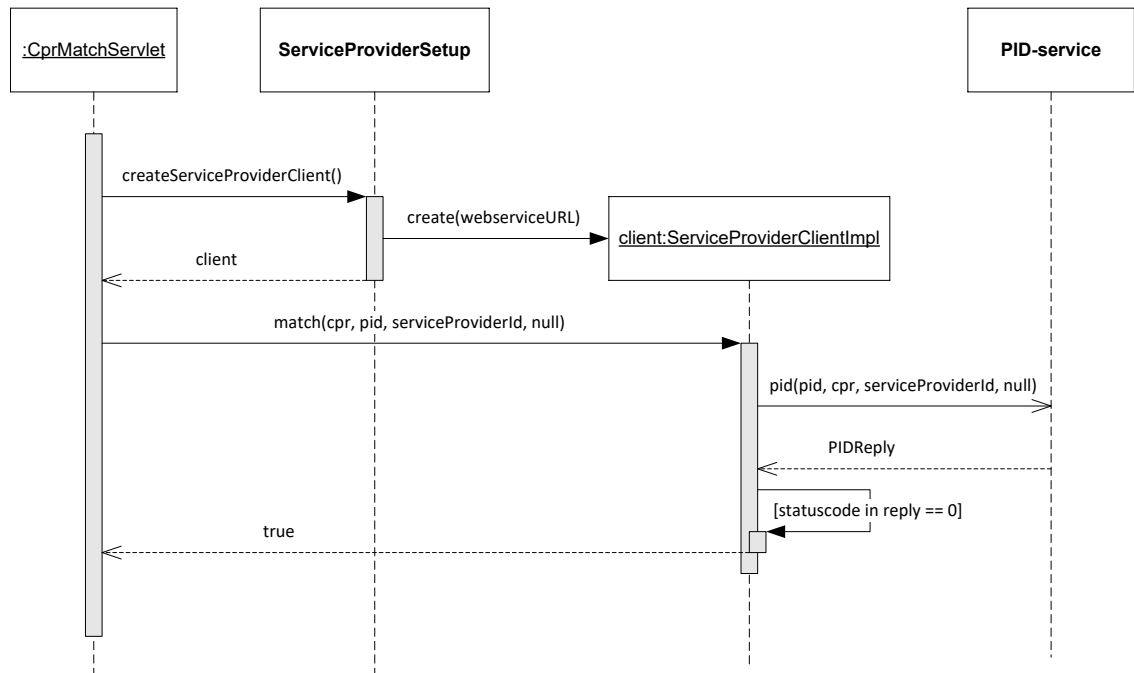
Figure 10-2 – Sequence diagram - CprMatchServlet performs a successful match call

As part of the login procedure at a Service Provider the user can be asked for his cpr-number. The Service Provider can then use the PID-CPR service for validating that the cpr-number and PID (retrieved during login) matches using a match call. Figure 10-2 depicts how the CprMatchServlet class performs a match call using the ServiceProviderClient class. The ServiceProviderClient is a webservice client in OOAPI. It can be used for calling the PID-CPR service. CprMathServlet uses the ServiceProviderSetup class for instantiating a ServiceProviderClient. The ServiceProviderSetup can furthermore be used to control which environment to execute in and to configure which revocation checker to use. Three revocation checkers are available in the OOAPI provided by Nets DanID:

- PartitionedCrlRevocationChecker – Retrieves the partial revocation list from CRL LDAP to perform the revocation check.

The certificate holds information about which partial revocation list it belongs to.

- FullCrlRevocationChecker – Retrieves the full revocation list from CRL LDAP to perform the revocation check. The FullCrlRevocationChecker caches the full revocation list. The maximum duration of the revocation list in the cache can be controlled by setting the "crl.cache.timeout.http" property in the ooapi.properties file.

- OCSPCertificateRevocationChecker – Performs revocation check by calling the OCSP system.

10.1.3 SignHandler

The SignHandler class offers a method of validating the output from the client against a given agreement text. A successful validation of signing data (i.e. a call which returns *true*) ensures that*:*

- The signature contained in the signing data is valid.

- The certificate contained in the signing data is valid, i.e. that the certificate has not expired or been revoked. By checking the returned status, you can see what state the certificate is in.

10.1.4 Example of web application in Java

As an example of how the OOAPI is used, Nets DanID has developed a simple web application for processing logon and signing using NemID. The application is built upon three scenarios:

- NemID Private (scenario 1)
- NemID Business (scenario 2)
- NemID Private and Business (scenario 3)

Each of these scenarios contain examples of logon and signing with code card and OCES code file.

The web application also contains a layout (interaction design) which Nets DanID recommends for the incorporation of NemID logon and signing.

Below, you will find a short introduction to the structure of the application. For further information, refer to Javadoc and the source code.

> The minimum requirement to compile the application is to have Maven version 2.1.1 installed.

Structure of the application

The application is structured as follows:

```
tuexample
        src/main/java                   Java classes
        src/main/resources              Other program resources
        src/main/webapp                 Web files
                resources               Stylesheets and JavaScript
                variant1                Scenario for NemID Private
                variant2                Scenario for NemID Business
                variant3                Scenario for NemID Private and Business
                extras                  Misc. Setup and PID lookup
                WEB-INF/web.xml          Configuration of web application
                *.jsp                   Web pages
        pom.xml  Maven POM
```

The idea is that if you are only interested in scenario 2, you can ignore the contents of the variant1 and variant3 folders.

Generation of Javadoc

To generate Javadoc, you must run the command:

```
mvn javadoc:javadoc
```

from the command line.

The generated Javadoc can be found in **target/site/apidocs**.

Execution using Jetty

To use Jetty to execute the application, you must run the command:

```
mvn jetty:run
```

from the command line.

Following this, the application can be found at
http://localhost:8082/tuexample.

Execution using Tomcat

To use a pre-installed Tomcat to execute the application, you must run the command:

```
mvn install
```

from the command line.

The application is thus packaged as a *war* file.

Next, copy **target/tuexample.war** to **<tomcat-dir>/webapps**.

The application can now be found at http://localhost:8080/tuexample
(or another port, depending on how Tomcat has been configured).

Example of logon

In each of the folders variant1, variant2, and variant3, there is a sub-folder called "restricted". The contents of this folder can only be accessed after a logon. If you attempt to access a page within this without being logged on, the LogonFilter will direct you to the logon page.

The redirection to the logon page is set up in web.xml in the following way (for scenario 1):

```
<filter>
  <filter-name>variant1SecurityFilter</filter-name>
  <filter-
class>dk.certifikat.tuexample.variant1.LogonFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>variant1SecurityFilter</filter-name>
  <url-pattern>/variant1/restricted/*</url-pattern>
</filter-mapping>
```

The LogonFilter class itself is composed of AbstractLogonFilter (which is common across the three scenarios) and a sub-class in each variant package. AbstractLogonFilter takes care of the overall control with the log-in check, while each sub-class takes care of redirecting to the specific logon page. Also, each sub-class makes sure that the right kind of certificate is used.

The logon page itself is composed of two tabs, e.g. for log-in with code card or code file. By default, the user is directed to the tab for logon with a code card, but on each page it is possible to set a cookie so that the system remembers which tab is preferred. Therefore, AbstractLogonFilter checks the cookie "preferredLogin" in order to decide which tab the user should be redirected to.

No matter which logon method the user chooses, the NemID client will send the result to the LogonServlet when the user clicks **OK**. LogonServlet is set up in web.xml as follows (for scenario 1):

```
<servlet>
  <servlet-name>variant1LogonServlet</servlet-name>
  <servlet-
class>dk.certifikat.tuexample.variant1.LogonServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>variant1LogonServlet</servlet-name>
  <url-pattern>/variant1/logon.html</url-pattern>
</servlet-mapping>
```

As is the case with the LogonFilter class, LogonServlet is composed of AbstractLogonServlet (which is common across all three scenarios) and a sub-class in each variant package.

It is the responsibility of LogonServlet to check that the user has logged in with a valid certificate, and that the certificate is of the expected type.

The LogonServlet servlet uses the Security Package's LogonHandler to validate the logon data and to extract a PID/RID number from the logon data. When the LogonHandler class has returned a PID/RID number, this is saved in the session:

```
@Override
protected void logon(HttpServletRequest req, HttpServletResponse
resp, CertificateAndStatus certificateAndStatus) throws IOException,
ServletException {
    HttpSession httpSession = req.getSession();
    if (isPoces(certificateAndStatus)) {
        String pid = ((PocesCertificate)
certificateAndStatus.getCertificate()).getPid();
        httpSession.setAttribute(KEY_PID, pid);
        httpSession.setAttribute(KEY_LOGGED_IN, Boolean.TRUE);

resp.sendRedirect(req.getContextPath()+"/variant1/restricted/kvitter
ing.jsp");
    } else {
        …
    }
}
```

This example of a web application is very simple. In a real web application, it might be expected that the PID number, for example, would be used to check whether the user with the given PID number has access to the restricted area. The application would then be able to look up the PID number in a local user table and display a text showing who is logged on (e.g. "Joe Bloggs is logged on").

The URL *logout.html* hits the LogoutServlet servlet, which logs out the user and forwards him to the front page of the given scenario. The user is logged out by removing the key from the HttpSession object that specifies that the user is logged in.

Example of signing

Signing consist, like logon, of several tabs. Here, you can sign with e.g. code card or code file. Based on the "preferredLogin" cookie (see under the example of logon), the relevant tab is shown to the user.

The SignServlet servlet, which, like LogonFilter and LogonServlet, consists of an AbstractSignServlet class and a scenario-specific sub-class, receives the result from the given NemID client and validates that the used certificate is valid, that the certificate is of the expected type, and that the correct text has been signed. For the latter check, the agreement text is placed as an attribute on the session.

SignServlet uses SignHandler to decide whether the signature data is correct. Depending on the outcome, the user is redirected to either a success page or an error page.

10.1.5 Example of web application in .NET

As an example of how OOAPI.NET is used, Nets DanID has developed a simple web application for processing logon and signing using NemID. The application is built upon three scenarios:

- NemID Private (scenario 1)

- NemID Business (scenario 2)

- NemID Private and Business (scenario 3)

Each of these scenarios contain examples of logon and signing with code card and with an OCES code file.

The web application also contains a layout (interaction design) which Nets DanID recommends for the incorporation of NemID logon and signing.

Below, you will find a short introduction to the structure of the application. For further information, refer to the documentation files and the source code.

> For compiling the project, you need at least .NET 3.5.

Structure of the application

The application is structured as follows:

```
tuexample.net
   include/     Files which are included in the HTML pages
```

```
variant1/     Scenario for NemID Private
variant2/     Scenario for NemID Business
variant3/     Scenario for NemID Private and Business
Properties/   Contains assembly.cs
resources/    js, css, and image files
tuexample/    Generation of challenge and error handling
*.aspx Webpages
Web.config    Web application configuration
```

Generation of documentation

For generating ndoc, please read the readme file:

Ooapi.net/Docs/README.txt

Execution in Visual Studio

For the execution in Visual Studio of tuexample.net, please consult the readme file:

Ooapi.net/Docs/README.txt

Setup of the web application

Configuration of the demo application is stored within web.config file appsSettings section and includes information for: NemID CodeFile client iframe source url, OTP iframe source URL, NemID target environment name (used to by org.openoces.ooapi.utils.Properties), location of a PFX file containing VOCES certificate, service provider IDs for RID/PID services and application name. Note, that password for the PFX certificate stored in encrypted form within secureAppSettings section.

10.1.6 Validation of CPR numbers

The CprMatchServlet class implements an example of how to validate the link between the user's certificate and CPR number in calls to the PID/RID-CPR web service.

```
/* validates link between CPR number and PID number by calling the
PID.CPR web service */
private boolean validateCPR(String cpr, String pid) {
  if (cpr != null) {
    ServiceProviderClientImpl pidCprService =
ServiceProviderClientImpl.createForTestEnv();
    return pidCprService.match(cpr, pid, "44");
  } else {
    return false;
  }
}
```

Calls to this service require you to authenticate yourself to it, both by a

company certificate (VOCES) and by providing a Service Provider ID (called a "SPID").

The Service Provider ID (SPID) is the "44" indicated in the code above. In addition, you are also required to configure a *truststore* in order to establish the SSL connection to the service.

The configuration of these two is given in the *pidclientsecurity.xml* file, where *wsclientkeystore.jks* corresponds to the company certificate and *wsclienttruststore.jks* corresponds to the required truststore.

The company certificate (VOCES) is unique to each Service Provider, while *wsclienttruststore.jks* is static.

11 Security Guidelines

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

11.1 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 users PC.

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

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 web-site in order to eavesdrop or tamper.

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

https://developer.mozilla.org/en-US/docs/Security/HTTP_Strict_Transport_Security

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

The Open Web Application Security Project (OWASP) lists a few more headers at https://www.owasp.org/index.php/List_of_useful_HTTP_headers, which should also be considered and evaluated.

12 Integration with the NemID CodeFile client

NemID CodeFile client is used for authentication and signing operations by customers who have OCES certificates stored locally as a code file on a file system or on NemID hardware. Operations using code card (OTP) are not possible with NemID CodeFile client.

NemID CodeFile client consists of:

- an iframe, which is responsible for embedding web UI. The page can contain only one NemID CodeFile client iframe

- a native application, which is based on OpenSign APIs

- a browser plugin, which enables communication between iframe and native application

NemID CodeFile client response format is XMLDsig. It is fully compatible with the NemID JavaScript clients response message format.

In order to instantiate NemID CodeFile client, the following code must be included on service providers page:

- A script tag with id=codefile_parameters containing client parameters in JSON format

- An iframe tag with id=codefile_iframe

- Java script for communication with the iframe using web messaging.

- A form tag with id=signedForm for encapsulation of response data from NemID CodeFile client to the service providers response handler.

This is similar to integration with NemID JavaScript client, described previously in the document.

All of the above can be generated with a help of CodeFileClientGenerator class, available within TUExample demo application, which provides a reference implementation for integration with both NemID CodeFile and NemID JavaScript clients.

If the NemID native application is not available on end users computer, the NemID CodeFile client will guide the end user to installation or fall back to the legacy OpenSign applet based implementation by dynamically injecting necessary HTML for applet rendering.

12.1 Parameters

List of parameters accepted by the NemID CodeFile client. Parameter names and values are case sensitive.

Name	Madatory	Encoding	Description
CLIENTFLOW	Y	None	Flow type. Values: login, sign
SP_CERT	Y	Base64	Service provider VOCES certificate issued by Nets-DanID certificate authority in DER format.
TIMESTAMP	Y	Base64	Current time when generating parameters. The timestamp must be supplied as the number of milliseconds since 1970-01-01 00:00:00 or as a formatted string. Examples: 2013-12-17 13:33:47+0100 1395819294069
ADDITIONAL_PARAMS	N	Base64	Supplementary parameters. Semicolon (;) separated key value pairs

REQUESTISSUER	Y	Base64	The service provider "Friendly Name" presented to users in logon flows.
LANGUAGE	N	None	Language. Values: da (default), en.
ORIGIN	Y	Base64	URL of the service provider domain. Format: https://<hostname>:<port> The <port> part must only be defined, if the service provider service is not using standard port-number.
PARAMS_DIGEST	Y	Base64	SHA-256 digest of the normalized parameters.
DIGEST_SIGNATURE	Y	Base64	RSA signature of the calculated parameter digest.
SIGN_PROPERTIES	N	None	XMLDSIG properties to be included in signed response. One or more name=value pairs separated by semicolons (;). Used for transporting challenge.
SIGNTEXT	Y*	Base64	Value of the text to be signed
SIGNTEXT_FORMAT	Y*	Base64	Format of the text. Values:text, html, xml, pdf
SIGNTEXT_TRANSFORMATION	N*	Base64	XSLT file. Required when SIGNTEXT_FORMAT=xml

*) – relevant for signing flow only

ADDITIONAL_PARAMS parameter is a container for supplementary parameters:

- subjectdnfilter parameter. It is used to filter certificates made available to the end user for a logon/signing. Provides ability to filter certificates based on their subject name. To show only POCES certificates, the filter should be set to UElEOg== (base64 representation of "PID:"), only MOCES – UklEOg== (base64 representation of "RID:"). Filter should be left blank if no filtering is required (default).

- issuerdnfilter parameter. It is used to filter certificates made available to the end user for a logon/signing. Provides ability to filter certificates based on a certificate issuer name. By default, only certificates issued by OCES CAs (both test & production) will be made available for the end user, while certificates issued by any other (non OCES) CAs will be excluded from the list. Examples:

Value	Meaning
VFJVU1QyNDA4IFN5c3RlbXRlc3Q=	Base64 value corresponding to "TRUST2408 Systemtest". Only certificates issued by OCES test

	CAs will be included.
VFJVU1QyNDA4IE9DRVM=	Base64 value corresponding to "TRUST2408 OCES". Only certificates issued by OCES productions CAs will be included.
VFJVU1QyNDA4	Base64 value corresponding to "TRUST2408". Only certificates issued by OCES CAs (production & test) will be included. The default.

- signtext.uri, signtext.uri, signtext.uri.hash.value parameters. Used when signing a PDF with OpenSign Applet.

- attachments parameter. Used when signing with OpenSign Applet. Enables attachment of secondary documents to the main one.

12.2   Logs

There are three types of CodeFile logs.

12.2.1 CodeFile native application logs

Location: <USER-HOME>/.oces/

Logs are outputted to nos.log file. Create an empty file called nos.debug in the same directory to activate verbose logging.

12.2.2 CodeFile Internet Explorer extension logs

Location: <LOCALAPPDATA>/Temp/Low/ (<LOCALAPPDATA> usually resolves to <USER-HOME>/AppData/Local)

Logs are outputted to NemidNoeglefilsprogram.log file. Create an empty file called NemidNoeglefilsprogram.allowdebug.txt within the same directory to activate verbose logging. Verbose logs are written to a different file called NemidNoeglefilsprogram.debug.log.

12.2.3 CodeFile Chrome extension, Firefox extension and iframe logs

Location: browser console.

13 Standards and algorithms

This chapter provides an overview of the standards that are used in the rest of the document, and thus functions as a list of prerequisites for adopting DanID. Appendix A contains further references to the standards.

13.1 XMLDSig

XMLDSig (XML Signature Syntax and Processing) is used by SAML to create and verify signatures embedded in messages.

The XMLDSig standard has been augmented by XMLEnc and RFC 4051. Both expand on the list of algorithms that an XMLDSig implementation should support. Most importantly, RFC 4051 adds the option of using SHA-256 and SHA-512 when signing.

13.2 XMLEnc

XMLEnc expands on XMLDSig to add support for encrypted elements, cipher data and various key transport and identification methods. Among the algorithms that XMLEnc adds are SHA-256 and SHA-512.

13.3 Cryptographic algorithms

This following lists the cryptographic algorithms used during the authentication and signing flows

Digests	SHA-256
Signatures	RSA with 2048 bit keys
Encrypted assertion	AES 256 in CBC mode with ISO10126 padding
Transport key encryption	RSA PKCS1

ISO 10126 padding uses the last byte of the decrypted cipher text to indicate the length of the padding block. The remaining bytes in the padding block are random. The resulting byte array should be trimmed accordingly upon decryption.

A. References

[CORS]	Cross-origin Resource Sharing http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
[Config-doc]	Configuration and Setup, v. 1.16 http://www.nets.eu/dk-da/Service/kundeservice/nemid-tu/NemID-tjenesteudbyderpakken-okt-2014/Documents/Configuration%20and%20setup.pdf
[JSON]	JavaScript Object Notation http://www.ietf.org/rfc/rfc4627.txt http://www.json.org/
[PKCS1]	PKCS #1: RSA Cryptography Specifications 2.0 http://tools.ietf.org/html/rfc2437#page-13
[RFC 4051]	Additional XML Security Uniform Resource Identifiers (URIs) http://www.ietf.org/rfc/rfc4051.txt
[SOP]	Same Origin Policy http://en.wikipedia.org/wiki/Same_origin_policy
[SP-docs]	Service Provider documentation package English: http://nets.eu/sp-package Danish: http://nets.eu/tu-pakke
[Web Messaging]	HTML5 Web Messaging http://www.w3.org/TR/webmessaging/
[XDRO]	XDomainRequest object http://msdn.microsoft.com/en-us/library/ie/cc288060(v=vs.85).aspx http://blogs.msdn.com/b/ieinternals/archive/2010/05/13/xdomainrequest-restrictions-limitations-and-workarounds.aspx
[XMLENC]	XML Encryption Syntax and Processing http://www.w3.org/TR/xmlenc-core/
[XMLDSIG]	XML Signature Syntax and Processing (Second Edition) http://www.w3.org/TR/xmldsig-core/
[Mobile-doc]	"NemID JavaScript client Integration for Mobile Applications" Can be found under "Dokumentation" in **[SP-docs]**
[Mobile-code]	SP example for iOS and Android Can be found under "Kildekode" in **[SP-docs]**
[LSS]	https://www.lss-for-nemid.dk/

B. Deprecated or renamed parameters

The following table contains a list of parameters that have been removed or renamed.

ALWAYS_EMBEDDED
APPLETLOG_ENABLED
attachments
DANID
EXTRA_FILE_NAME_PARAM
locale
ocesprovider
opensign.alerturi
opensign.canceluri
opensign.cookiecount
opensign.cookie#name
opensign.cookie#value
opensign.doappletrequest
opensign.doappletrequestonmac
opensign.erroruri
opensign.formdata.count
opensign.formdata#name
opensign.formdate#value
opensign.message.name
opensign.result.name
opensign.verifieddokuri
opensign.verifieruri
Paramsdigest   replaced by PARAMS_DIGEST
requestIssuerId
setFocusAfterAppletFunction
setFocusBeforeAppletFunction
ServerUrlPrefix
Signeddigest   replaced by DIGEST_SIGNATURE
Signproperties   replaced by SIGN_PROPERTIES
signText.chunk
sso
TWO_FACTOR   replaced by CLIENTFLOW
USESHA1   SHA256 is the only supported hash algorithm for the NemID JavaScript Client
windowDecorated
ZIP_BASE_URL
ZIP_FILE_ALIAS   replaced by CLIENTFLOW

C. PDF Whitelist

The following is the complete PDF whitelist:

PDF Types:

/FontDescriptor

/Font

/Metadata

PDF keys:

/Encoding

/ExtGState

/ColorSpace

/Pattern

/Shading

/XObject

/ProcSet

/Properties

/BaseFont

/Name

/Dests

/Dest

/Info

/Font

/Differences

PDF 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

/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
/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

/Diamond

/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

/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

/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

/ForComment

/ForPublicRelease

/Form

/FormEx

/FormType

/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
/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
/LZWDecode
/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

/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

/SoftLight

/Sold

/Solid

/Solidities

/Sort

/SoundActions

/SpaceAfter

/SpaceBefore

/Span

/SpawnTemplate

/SpotFunction

/Square

/Squiggly

/St

/Stamp

/Standard

/Start

/StartIndent

/State

/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

/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

/VisiblePages

/W

/W2

/WMode

/Warichu

/Watermark

/WhitePoint

```
/Width
/Width2
/Widths
/WinAnsiEncoding
/WritingMode
/X
/XFAResources
/XHeight
/XML
/XObject
/XRef
/XRefStm
/XStep
/XYZ
/Xsquare
/Y
/YStep
/Yellow
/Ysquare
/ZapfDingbats
/Zoom
/adbe.pkcs7.detached
/adbe.pkcs7.sha1
/adbe.x509.rsa_sha1
/ca
/cb
/checked
/max
/min
/neutral
/null
/off
/on
/op
/pb
/rb
/tv
```

Microsoft Office elements:

/Workbook

/Textbox

/Endnote

/Worksheet

/Macrosheet

/Annotation

/Dialogsheet

/Chartsheet

/Diagram

/Footnote

/Chart

/Slide

/InlineShape

/Artifact

/Figure

/Formula

A SaaS APPLICATION ON TEXT MESSAGING SOLUTIONS – A ColdFusion Case Study

Executive Summary:

A 100% open rate, 98% read rate and 90% response rate is an enviable figure where digital communication is considered. If you haven't guessed it already the above statistics are for text messages. The proliferation of mobile phones all over the world (it is already 100% in the USA) has led to many brands opting to include text messaging into their digital marketing strategy. Yet another statistics states that 43% of mobile advertisements prompted brand recall. These statistics confirm the fact that messages delivered via mobile text advertising are more personal and timelier than any other advertising medium. The reach of mobile text advertising is even more owing to the ubiquity of mobile phones as a constant companion in our lives.

And this is precisely our client's business plan. Developing a SaaS application on text messaging solutions that would help various advertisers effectively reach their target audience with personal messages in a timely manner was the top most priority for them. Additionally this would also help build a consumer database which could be analyzed further and contacted accordingly. As a software services provider they wanted to leverage on the software technologies readily available to communicate better with the target market.

Apparently when the client contacted Mindfire, they had already been through a negative experience with the previous software vendor. However, the partnership with Mindfire proved to be a fruitful one and now we are the primary software services provider for the client. They have also expressed interest in Mindfire's services when they would want to customize the platform, white label the product and aggressively market it to other domains.

About the Client:

Client: Mobile Messaging Services Provider

Industry: Marketing and Advertising

Location: USA

Technologies Used:

ColdFusion10, SQL Server 2008, Bootstrap, ChartJS, BaseCamp, jUnit

Business Situation:

Our client wanted to stay competitive in the Text Messaging Services Industry that has a significant share in the Marketing and Advertising campaigns in USA and elsewhere in the world. However, when they approached Mindfire Solutions their Code was not in the best condition and they had to restart the servers on a daily basis and there were times when the messaging broadcast would fail randomly.

In this scenario, Mindfire had its task cut and proceeded with a modular approach giving more weightage to the more pressing issues first.

The Mindfire Solution:

The primary customers of our client were Advertisers who can get distributed Shortcode or they can own a Shortcode. The Advertiser can set up campaigns for Restaurants, Comedy Clubs, College, Real Estate etc. They can also set Zip Code location processing for a scheduled Broadcast. The main purpose of the Campaigns was to give real time notifications to a user about an offer going on, Coupons & information about the Advertiser's shop location primarily through Broadcasts or through Response Messages. The Advertisers also get access to the subscribers' Name, DOB (date of birth), email, Zip etc. which helps them to provide the best services.

At Mindfire Solutions, we analyzed the code base which had flaws and did code optimization in every corner of the application. Our developers wrote approximately 75 stored procedures which had a capacity to give 5 to 7 resultsets. We used Bootstrap, ChartJS for Admin dashboard without dependencies on the Base CSS. Most of the queries were replaced by the stored procedures so that Database hit was managed. The performance of the application improved considerably without having the client to restart or suffer from a system hanging up while broadcasting.

We faced a challenge for new API integration without support from Service providers. Our developers worked on a new API for the client which in turn enabled the Advertiser to send test to N numbers, N different messages in one to one relation. This API was set up using CF Scheduled services and Caching query result. The Stats, Growth, Summary Reports, Messages Logs and Advertiser Dashboard where set up so Advertisers and Our Client have a clear picture on Application and Campaigns. Surprise Redemption Code campaign was set up with having 2 way communications with Advertiser's application, which went on for several months with success.

Fetching DOB data collection from Subscriber for giving special offers on birthdays was set up, by using a list of Regular Expressions to accept any format. The data was achieved with 80% success. A robust structure was established for setting up the campaigns so that the Advertiser can limit the number of outgoing messages, collect the data and send emails in real time. New Mail Chimp, Custom JSON API was set for sending real time data to Advertisers. Our developers also converted HTTP to SMPP protocol because the providers had a lot to change on their side. The application stopped after successful transfer of the Shortcodes. We also applied every New CTIA changes for sending of Messages to Subscribers.

The application had a unique pass-thru facility to the Advertisers which notifies them as soon as they get Subscriptions or any information for Campaigns. The various tracking reports were installed to get the Subscribers List, Growth of Subscribers, Stats given by subscriber and Summary Report for getting

response messages that went out from Application for Subscriber messages. The application also had few API Listeners open so that the Advertiser can send messages without logging in or can manage broadcast at their end. Base camp was used for Bug tracking and jUnit tool was used by client end for testing.

Architecture Diagram:

Advertisers

Advertiser can own one or more Designated Shortcode or one or more Shared Shortcode

Shortcode

Shortcode can have any number of Unique Keyword or Alias Keyword

Keywords

Keyword can have only one Campaign, there the Response Message and Other

Campaigns

Under Campaigns Advertisers can create Groups as many they need

Groups

Subscribers can be Directly Under Campaigns

Subscribers can be Under Groups or Can directly under Campaigns

Subscribers

Fig 1: The above figure shows the architecture diagram of the Text Messaging Solution

How the System works?

MT: Mobile Termination

Advertiser → **Text Message App** → **Service Provider** → **Subscriber**

Pass Thru:
1. **XML Post Pass Thru**
2. **FishBowl Pass Thru**
3. **SalesForce Pass Thru**
4. **MailChimp**
5. **Custom JSON**

MO: Mobile Origination

Fig 2: The above figure shows the WorkFlow image of the Text Messaging Solution

There were 3 processes which worked behind the application:

1. **MO(Mobile Origination):** Here the Messages is received by Subscribers. The message is processed and marked valid or invalid

2. **MT(Mobile Termination):** Here the Message Response from Application to Subscribers and Broadcast set by Advertisers were handled.

3. **Pass Thru:** There were many kinds of Pass Thru in the Application. These are used to indicate the Advertisers in Real time about data received.

Screenshots:

Following are some of the screenshots of various sections of the application:

Screenshot 1: User Dashboard signifying the various metrics of the subscriber base

Active Short Codes Database: Edit Code

« Return to List

Code: 10936
Site:
Owner:
☐ Make this short code searchable by website users.
Gateway:
Max Message Length: 160
Accout Name (M365 only):
Password (M365 only):
Login Base 64 (M365 only):

☐ Enable pass-thru processing for non-keyword traffic on this short code.
Pass-Thru URL:
Username:
Password:
☐ Enable pass-thru acknowledgements
☐ Enable numeric blob processing

System Messages   **Variable Quick-reference**
Opt-In: [campaignDesc]: U will rcv [services]   134 / 160°

[campaignDesc]: U will rcv
[services].   Msg
[frequency]. NoPurchaseRqd.
End: txt STOP. Help: txt HELP.
Msg&DataRatesMayApply

Screenshot 2: Short code database for a particular Advertiser

Message Log

Group	Short Code	Source	Status
Recent ▼	--- ▼	▼	--- ▼

Carriers	Mobile #	Message	Start Date	End Date
--- ▼			9 ▼ / 11 ▼ / 2015 ▼	9 ▼ / 14 ▼ / 2015 ▼
			12 ▼ : 00 ▼ : 00 ▼ am ▼	11 ▼ : 59 ▼ : 59 ▼ pm ▼

Submit

Click here to clear the message log.
1

CSC	Source	Destination	Message	Rcv Time	Snd Time	Status
68247	5 T-Mobile			09/13/2015 9:25:01 PM	09/13/2015 9:25:01 PM	Texted Invalid
68247	5	5		09/13/2015 9:24:46 PM	09/13/2015 9:24:46 PM	Texted Invalid
68247	5	5		09/13/2015 9:24:43 PM	09/13/2015 9:24:43 PM	Texted Invalid
68247	5 T-Mobile			09/13/2015 9:24:14 PM	09/13/2015 9:24:14 PM	Texted Invalid
68247	3 T-Mobile	57847		09/13/2015 9:18:02 PM	09/13/2015 9:18:03 PM	Texted
68247	3 T-Mobile			09/13/2015 9:17:22 PM	09/13/2015 9:17:22 PM	Texted Invalid
68247	75757 Verizon			09/13/2015 9:06:48 PM	09/13/2015 9:06:49 PM	Texted Invalid
68247	1			09/13/2015	09/13/2015	Texted Invalid

Screenshot 3: Message Log for the outbound messages and their corresponding status.

Keywords Database: Edit a Keyword

« Back to List

Basic Setup
Keyword: 1
Aliases:
                Separate aliases with ','
Subscriber:
Campaign Description:
Service Description: shipment confirmation
Short Code:
Service Type: Basic TXTPak
Purchase Date: 8 ▼ / 15 ▼ / 2015 ▼
Expiration Date: 12 ▼ / 31 ▼ / 2015 ▼
Send Text Messages Via: Standard Text Message ▼
Support Contact: Important Text Alert
Support Email:
Support URL:
Support Phone:
Custom HELP Message:

☐ For Marketing
☐ Send emails on remote broadcasts
☐ Use hyphenated tokenization
☑ Suppress CTIA MT
☐ Enable zip location lookup
☑ Exclude from texted system menus
☐ Exclude HELP and STOP messages from Sending Out
☐ Response Override for API Subscriptions

Screenshot 4: Keywords Database to increase effectiveness of the text messaging campaigns.

Future Relationship:

This has been one of our long running projects and each year the client was satisfied with the work done at Mindfire Solutions. They have also referred the services of Mindfire Solutions to their peers and that has been a rewarding experience. Not only have we added more clients but we also get to work on various platforms to meet the needs of the client and their customers.

We are also the primary software services provider to the client who is planning to white label the product as sell it to other organizations across industries.

Developer Speak:

"It was an awesome opportunity to work on this application. Needless to say, the learning curve never diminished. The challenges were a learning experience as huge real time issues were solved with CF and SQL."

- Ramakrishna Panni

mindfiresolutions
BURN IGNORANCE

Founded in 1999, Mindfire Solutions is an award-winning provider of software development and testing services to the global market with 650+ talented software engineers at 3 centers in India. For its people and its work, Mindfire has won coveted international awards such as Deloitte Technology Fast50 India Award 2013 and 2014, Dun & Bradstreet Fastest Growing SME 2013 Award, Red Herring Top 100 Asia Award and Zinnov GSPR 2014. Mindfire has been recognized with ISO 9001:2008 and ISO 27001:2005 certification, is a continuous member of NASSCOM, and has established a strong track record of 2000+ projects successfully delivered for 500+ technology clients.

© 2015 Mindfire Solutions | www.mindfiresolutions.com

Need help to create a text messaging solution? Reach out to us here:

Phone: 1-248-686-1424

Web: www.mindfiresolutions.com

Email: sales@mindfiresolutions.com