

csc gateway Manual & Reference

Juli 2023

intarsys GmbH

csc gateway Manual & Reference

Version 3.1

csc gateway for sign-me

intarsys GmbH
csc gateway Manual & Reference
Version 3.1

All rights reserved
© 2023 intarsys GmbH
www.intarsys.de

Preface

- Author and company

This book has been provided by different authors from the development staff of intarsys GmbH.

- Trademarks

Wherever possible and where the authors were aware of a trademark claim, such designations are marked as trademarks in this book.

csc is a trademark of cloud suite consortium.

sign-me is a trademark of Bundesdruckerei GmbH.

Sun, Java and JavaScript are trademarks of Oracle

- Who should read this book

This book provides both an overview of the product design and architecture and a manual for installation and usage.

So, this is the document for architects, developers and operators.

- Reviews and comments

We make constant efforts to improve our documentation and meet your requirements. Your comments are welcome as a valuable resource for us.

EMail support@intarsys.de

Website www.intarsys.de

Contents

Preface	5
▪ Author and company	5
▪ Trademarks	5
▪ Who should read this book	5
▪ Reviews and comments	5
Contents	7
Introduction	10
1. Changes	11
2. Installation	12
2.1 Overview	12
2.2 System requirements	12
2.2.1 Server	12
2.3 Documentation	13
2.4 Web Apps	13
2.4.1 Common	13
2.4.2 Web App "csc"	13
2.4.3 Web App "authorization"	14
2.4.4 Web App "authorization-admin"	14
2.5 Version check server	14
2.5.1 In the WEB-INF directory	14
2.5.2 In the log	15
3. Design details	16
3.1 Overview	16
3.2 Signing	16
3.2.1 Sequence diagram	17
3.3 Sealing	17
3.3.1 Sequence diagram	20
3.4 Accounting	20

3.4.1	account_token	20
3.4.2	sign-me service endpoint	21
3.4.3	sign-me partner	21
3.4.4	sign-me account	21
3.4.5	Account derivation	21
3.5	Credential enumeration & info	21
3.6	Credential authorization	21
3.7	Multi signature transaction	22
3.8	V.0 API mapping & support	22
3.8.1	Supported methods	22
3.8.2	Unsupported methods	22
3.8.3	/info	23
3.8.4	/auth/revoke	23
3.8.5	/oauth2/authorize	23
3.8.6	/credentials/list	23
3.8.7	/credentials/info	24
3.8.8	/signatures/signHash	26
3.9	V.1 API mapping & support	26
3.9.1	Supported methods	26
3.9.2	Unsupported methods	26
3.9.3	/info	27
3.9.4	/auth/revoke	27
3.9.5	/oauth2/authorize	27
3.9.6	/oauth2/token	27
3.9.7	/credentials/list	28
3.9.8	/credentials/info	28
3.9.9	/credentials/authorize	29
3.9.10	/signatures/signHash	29
3.10	Short time certificates	29
3.11	Enrollment	30
4.	Configuration	32
4.1	Overview	32
4.1.1	Configuration location	32
4.1.2	Built-in configuration	34
4.1.3	Custom property definition	34
4.1.4	Custom bean definition	35
4.1.5	Using profiles	35
4.1.6	String expansion syntax	36
4.2	authorization app properties	36
4.3	csc app properties	37
4.4	csc app deprecated properties	43
4.5	Registering sign-me accounts	44
4.6	Registering OAuth2 clients	48
4.7	Web application root	49

4.8	Logging	50
4.8.1	Override logging	50
4.8.2	Builtin logging	51
4.8.3	Log tweaks	52
4.9	Licenses	52
5.	Web apps	54
5.1	"csc" 54	
5.2	"authorization"	54
5.3	"authorization-admin"	54
5.3.1	UI	54
6.	Adobe Sign screen flow	56
7.	UI customization	64
7.1	Overview	64
7.2	authorization app	64
7.2.1	Consent pages	64
7.2.2	Error pages	65
7.2.3	Hints	65
7.3	csc app	65
8.	NLS	66
8.1	NLS resolving	66
8.2	Additional resource path	67
9.	Appendices	68
9.1	Cheat sheet	68
9.1.1	Windows locations	68
9.1.2	Linux locations	68
9.1.3	Property definitions	69
9.1.4	Bean definitions	69
9.1.5	Logging	69
9.1.6	Licenses	69
9.2	Test environment	70
10.	External References	71

Introduction

csc gateway implements a gateway from the csc remote signing API, to the Bundesdruckerei sign-me backend. Both version v.0.1 and version v.1.0 are supported using the respective URLs of the specification.

Using this gateway, a csc standard compliant client is able to create qualified electronic signatures on the sign-me backend.

This book contains all relevant architecture, design, implementation and operating information for the product.

1. Changes

mit,...	initial releases
mit, 17.3.2020	auth error pages improved; added jws.tokenValidity property added authRegistry.retentionTime property
mit, 17.11.2020	add property
mit, 3.1.2023	Support for per-client accounts
mit, 1.6.2023	Support for v.1

2. Installation

2.1 Overview

The product is shipped in two ZIP files,

- is-csc-signme-*.zip
- is-auth-signme-*.zip

containing all components and documentation.

The deployment artifact contains these directories

- webapps
All web applications, packaged as standard WAR files.
- doc
All documentation files.
- examples
All files provided for testing and demonstration purposes

2.2 System requirements

2.2.1 Server

2.2.1.1 Operating system

The server components are tested and released for

- Ubuntu Linux 16.04 or higher

2.2.1.2 System software

Java server runtime

The server components are developed in Java and require

- Java Runtime Environment 17

Servlet container Tomcat

The server components are deployed as "war" files. The server components are tested and released using

- Tomcat 9.0.x

2.2.1.3 Hardware

We recommend at least 1GB of dedicated memory on a low profile server machine.

2.3 Documentation

The documentation (including this document) for the complete system is in the ZIP file for is-csc-signme, folder "doc".

2.4 Web Apps

2.4.1 Common

The web applications are installed by copying the respective WAR files from the ZIP file "webapps" folder into the "webapps" directory of the servlet container.

For production use it is required to address the applications using TLS. You need either to configure the TLS termination for the servlet container or provide a TLS terminating proxy.

2.4.2 Web App "csc"

The "csc" web app, to be found in the "is-csc-signme" ZIP file publishes the csc interface for integration in clients like "Adobe Sign" or other compatible applications. On the implementation side, the sign-me remote signature API is used.

The application artifact "csc.war" can be found in "is-csc-signme" ZIP file, folder "webapps" and must be deployed in the "webapps" folder of the servlet container.

If this application and "authorization" are deployed in the same container with the default application context names, no further configuration is required to "connect" the two applications, the configuration defaults are sufficient for this case.

When the two applications are deployed in different containers, be sure to configure the property

- backend.authServer

2.4.3 Web App "authorization"

The "authorization" web app implements an OAuth2 service for use in combination with the "csc" implementation.

The application artifact "authorization.war" can be found in "is-auth-signme" ZIP file, folder "webapps" and must be deployed in the "webapps" folder of the servlet container.

If this application and "csc" are deployed in the same container with the default application context names, no further configuration is required to "connect" the two applications, the configuration defaults are sufficient for this case.

When the two applications are deployed in different containers, be sure to configure the property

- backend.cscServer

2.4.4 Web App "authorization-admin"

The "authorization-admin" web app implements administrative features.

The application artifact "authorization-admin.war" can be found in "is-auth-signme" ZIP file, folder "webapps".

This application is not required for production use and should not be public available.

2.5 Version check server

2.5.1 In the WEB-INF directory

In cases you need to physically check the installation version on the deployed server application, you can look up the "<webapp>/WEB-INF/version.txt". It contains tags that describe the artifact version

```
version=8.0.0  
build=321  
timestamp=Mon Apr 23 14:15:54 CEST 2018
```

2.5.2 In the log

The log file is the most important source of information for troubleshooting an installation. You can find the version of every component contained in the deployment near the beginning of the log file

```
[12:20:49.554][I][d.i.spring.tools][main][[] version info:  
[12:20:49.554][I][d.i.spring.tools][main][[] is-csc  
sign-me, 1.0.0  
[12:20:49.556][I][d.i.spring.tools][main][[] +- ASM (asm-  
5.2.jar), 5.2  
...
```

3. Design details

3.1 Overview

The csc specification provides itself a "framework" of protocols and conventions that are valid for remote signature creation.

This sign-me gateway implementation specific decisions are documented here.

Many of them are derived from current restrictions and implementation decisions on the existing applications

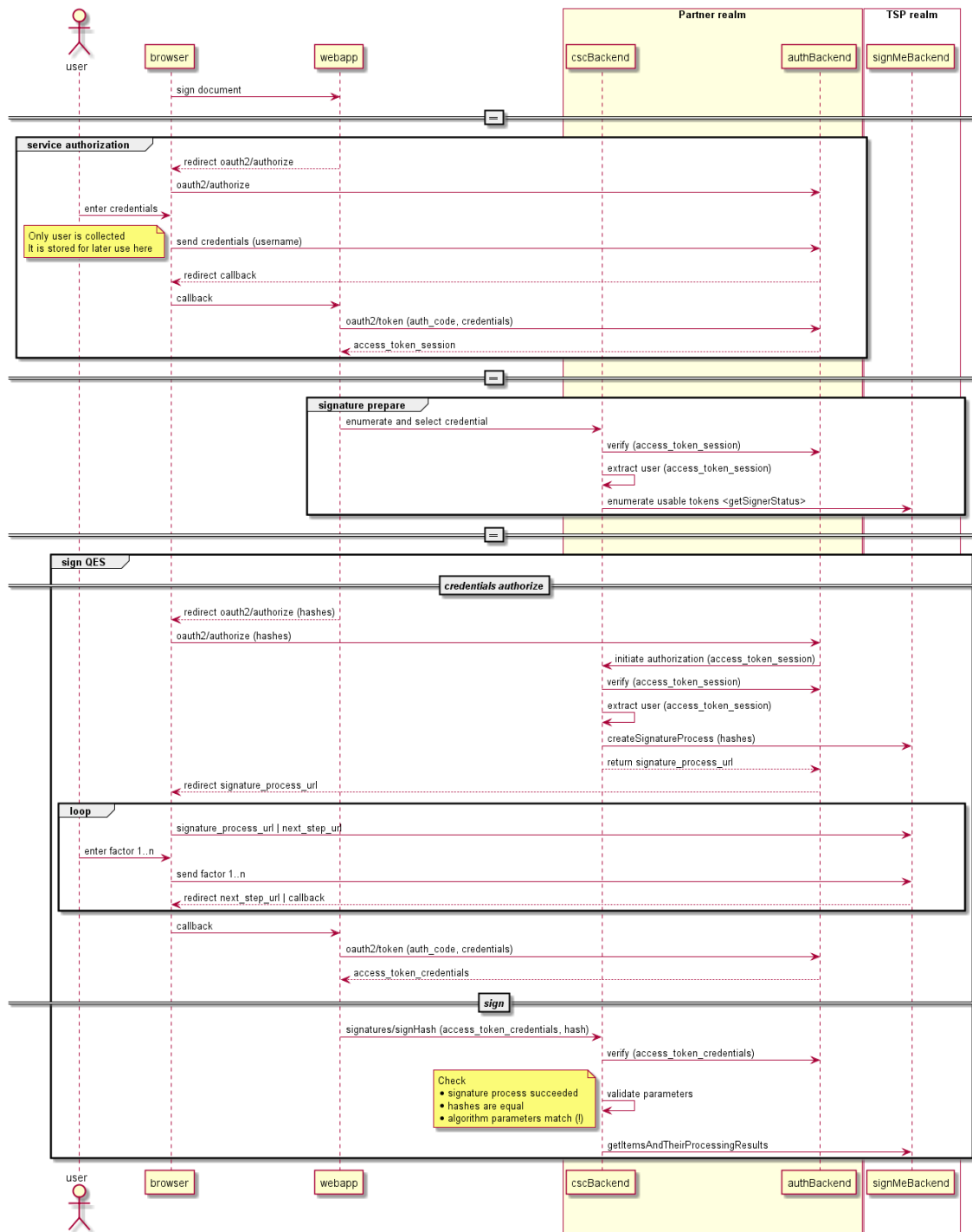
- sign-me remote signature service
- Adobe Sign client

3.2 Signing

Service authorization is supported using OAuth authorization code flow. As the "PARTNER" role is not allowed to explicitly authenticate a user but a "user based" scenario is required for the Adobe Sign integration, we collect upfront the username for further unauthenticated internal use (credentials enumeration). The authorized user is compared to the user parameter for later calls.

A two-factor authentication is delayed until credential authorization. Here, user consent is collected using sign-me UIs, implemented via OAuth authorization code flow again.

3.2.1 Sequence diagram



3.3 Sealing

The sealing process supported by Adobe is described in [1] and [2].

Sealing is supported based on the features

- OAuth2 client credentials flow for service authorization

- PIN authentication mode for the respective credential

Service authorization is supported via client credential flow for sealing. Be sure to allow “client_credentials” as a “grantTypes” member in your clients configuration.

Explicit PIN authentication is added to all sign-me tokens that support sealing.

Initially getting a sign-me sealing token is a process described in the respective D-Trust documentation. After this process you will receive a P12 file that contains an authentication/authorization token of the sealing process. This key must be deployed to a configurable folder (sign-me.seal.authenticationTokenFolder) that contains all authentication tokens for the CSC gateway.

The token itself has the following important properties

- The token is issued to an organization.
- The organization name is the alias for the token key
- The token has an additional CSM number
- Both organization name and CSM number are encoded in the token file name.
- Together organization and CSM number identify a token in the sign-me backend
- The P12 key has a PIN that must be used to sign the authentication challenge with the token key

The Adobe seal configuration requires the following properties

- OAuth client id for the account that is allowed to use the seal
- OAuth client secret for the account that is allowed to use the seal
- The credential id that identifies the seal
- The credential PIN that is required to authorize the seal

These properties are mapped to the sign-me part

- OAuth client & secret match an existing client configuration at the OAuth server
- The client configuration must support the grant type “client_credentials”
- The CSC gateway needs an account definition for this client
- The account must support the tokenCapability “SEALING_QUALIFIED”
- The account must contain an appropriate authentication token definition for the credential id.
- The credential id matches the sign-me authorization token based on the following rule:

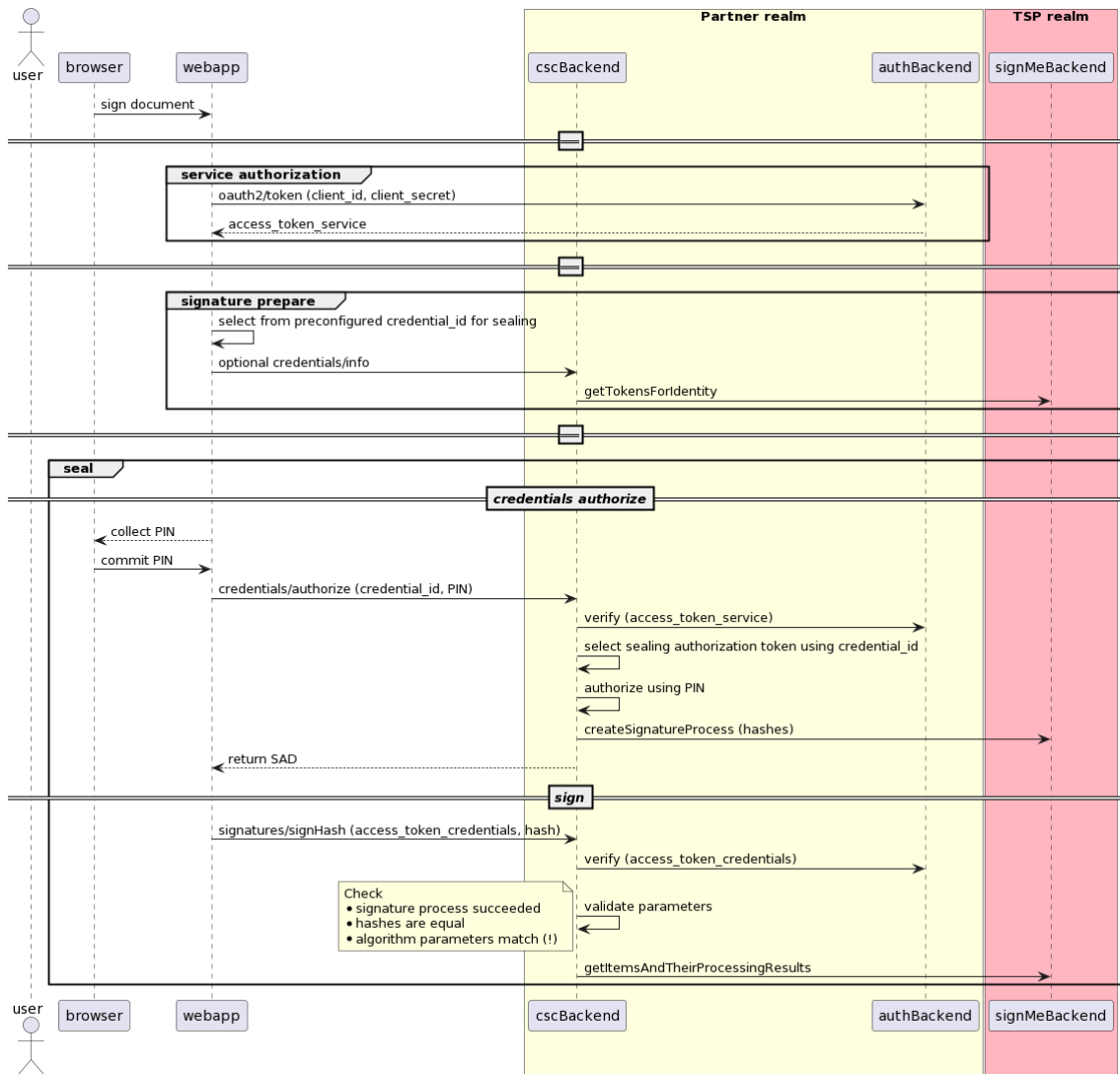
- There is an entry in the “authenticationTokens” property of the selected account with a matching “id” property (see “Registering sign-me accounts” section)
- All other properties are derived from this entry
- The credential PIN matches the sign-me authorization token PIN

To prevent PIN enumeration, the credential authorization step is guarded by a retry counter mechanism:

- Before each authorization, a “retry” property in a key-value store associated with the authentication token is incremented
- Authentication is performed
- After successful authentication, the “retry” property is reset
- If the number of unsuccessful authentication attempts exceeds the maximum retries (see configuration), the authentication is permanently locked. **Operator intervention is required to resolve this state!**

The key-value store is located at `${backend.keyValueStore}` in a file named “auth-“<tokenname>.json. This file may safely be deleted.

3.3.1 Sequence diagram



3.4 Accounting

3.4.1 account_token

The adobe csc client is able to send an "account_token" along with the service authorization via OAuth2. This token can be mapped to a service endpoint / partner access tuple.

The token is specified in v1 of the csc spec (but already available and usable in the v.0.1 implementation).

It is a signed JWT token, where the "sub" claim designates uniquely the originator of the signature task.

To receive this token requires prerequisite configuration on the adobe client side.

3.4.2 sign-me service endpoint

This is the HTTP(S) endpoint for the sign-me web services. Accessing the endpoint requires basic authentication.

3.4.3 sign-me partner

The csc gateway requires a dedicated partner account with the role "PARTNER".

3.4.4 sign-me account

This "account" is a virtual tuple containing

- The "sub" claim from the account_token
- The "client_id" from the OAuth2 authorization
- The sign-me service endpoint parameters
- The sign-me partner parameters

3.4.5 Account derivation

- accountId is derived from Adobe "account_token" (or null)
- clientId is derived from OAuth2 client_id (or null).
- if a sign-me account matching the accountId *and* the clientId is found, the corresponding service endpoint and partner parameters are used from this account. This way you can provide different parameter sets to different clients (with the same accountId),
- if a sign-me account without clientId matching the accountId is found, the corresponding service endpoint and partner parameters are used from this account for all clients.
- if no account is found, the lookup is retried with the accountId "default".
- If still no account is found, a "default" account is derived from the server properties (see properties)

3.5 Credential enumeration & info

The csc gateway collects information for the user that is determined in the service authentication. The implicit user context is supported only.

3.6 Credential authorization

Authorization is completely in the scope of sign-me.

For all credentials the "OAuth2Token" authorization mechanism is used.

While the authorization & signature step is split in two requests in the csc API design, sign-me only knows of a single "createSignatureProcess".

To map this asynchronous "createSignatureProcess" on a csc API **only** the credentials authorize via OAuth2 step is a valid candidate, as it is the only asynchronous mechanism supporting HTTP redirects.

This means that the sign-me signature creation is hidden behind the credential authorization, the signature call will only collect the result later on.

3.7 Multi signature transaction

The csc gateway supports multi signature transactions (according to backend constraints) based on a single signature call only.

Transactional calls are not supported.

3.8 V.0 API mapping & support

3.8.1 Supported methods

info	Collect statically configured attributes
auth/revoke	-
oauth2/*	Use private UIs and sign-me backend UIs for identification, authentication and authorization
credentials/list	The credentials associated with the authenticated user
credentials/info	Credential attributes
signatures/signHash	Sign hash values

3.8.2 Unsupported methods

auth/login	Only OAuth2 is supported auth/login with "BASIC" authorization exists per requirement of Adobe test beds with hard coded user
credentials/authorize	Only OAuth2 is supported
credentials/extendTransaction	not supported
credentials/sendOTP	Empty implementation for Adobe conformity purposes
signatures/timestamp	not supported

3.8.3 /info

specs	0.1.7.9
name	Bundesdruckerei sign-me
logo	https://www.bundesdruckerei.de/themes/custom/bs_bdr/images/bundesdruckerei-logo.png
region	DE
lang	de-DE
description	sign-me Signaturdienst der Bundesdruckerei
authType	oauth2code
oauth2	<configurable>
methods	auth/revoke oauth2/token credentials/list credentials/info signatures/signHash

3.8.4 /auth/revoke

Revocation is forwarded to the OAuth server.

The CSC implementation does not issue tokens of its own.

3.8.5 /oauth2/authorize

This endpoint is implemented in the "authorization" web app. A standard OAuth2 authorization flow is launched.

For "service" authorization, a "csc-gateway" private web page is published for collecting the sign-me username.

For "credential" authorization, the sign-me backend authentication page is used. Be aware that behind the scenes, this step implicitly creates a signature process at the sign-me backend.

3.8.6 /credentials/list

All usable tokens available for the service users are listed.

As "on-demand" certificates are not yet usable in the csc specification, all such tokens are filtered.

Tokens that do not provide algorithms supported by the Adobe Sign implementation are filtered.

3.8.7 /credentials/info

The token attributes are mapped to csc credential info attributes.

This process is straightforward for simple fields like "algorithm" or "length".

For other fields the following rules apply:

- As tokens that cannot be used are filtered, "status" is always "enabled".
- "authMode" is always "oauth2code"
- SCAL is always SCAL2 to enforce "hash" propagation upon authorization

3.8.7.1 CredentialInfo from basic Token

description	token.tokenTypeDescription
key/status	enabled
key/algo	RSA_PSS
key/len	2048
key/curve	-
cert/status	-
cert/certificates	cert chain
cert/issuerDN	
cert/serialNumber	
cert/subjectDN	
cert/validFrom	
cert/validTo	
authMode	oauth2code
SCAL	SCAL2
PIN/*	-
OTP/*	-
multisign	true
lang	

3.8.7.2 CredentialInfo from advanced Token

description	token.tokenTypeDescription
key/status	enabled
key/algo	RSA_PSS
key/len	2048
key/curve	-
cert/status	-
cert/certificates	cert chain
cert/issuerDN	
cert/serialNumber	
cert/subjectDN	
cert/validFrom	
cert/validTo	
authMode	oauth2code
SCAL	SCAL2
PIN/*	-
OTP/*	-
multisign	true
lang	

3.8.7.3 CredentialInfo from qualified Token

description	token.tokenTypeDescription
key/status	enabled
key/algo	RSA_PSS
key/len	2048
key/curve	-
cert/status	-
cert/certificates	cert chain
cert/issuerDN	
cert/serialNumber	
cert/subjectDN	
cert/validFrom	

cert/validTo	
authMode	oauth2code
SCAL	SCAL2
PIN/*	-
OTP/*	-
multisign	true
lang	

3.8.8 /signatures/signHash

This implementation only validates and collects the signature result within this method. The "impedance mismatch" between csc and sign-me does not allow for another mapping.

3.9 V.1 API mapping & support

3.9.1 Supported methods

info	Collect statically configured attributes
credentials/list	The credentials associated with the authenticated user
credentials/info	Credential attributes
credentials/authorize	Authorization for sealing tokens
signatures/signHash	Sign hash values

3.9.2 Unsupported methods

auth/login	auth/login with "BASIC" authorization exists per requirement of Adobe test beds with hard coded user
credentials/extendTransaction	not supported
credentials/sendOTP	Empty implementation for Adobe conformity purposes
signatures/timestamp	not supported

3.9.3 /info

Specs	1.0.3.0
name	Bundesdruckerei sign-me
logo	https://www.bundesdruckerei.de/themes/custom/bs_bdr/images/bundesdruckerei-logo.png
region	DE
lang	de-DE
description	sign-me Signaturdienst der Bundesdruckerei
authType	oauth2code, oauth2client
oauth2	<configurable>
methods	auth/revoke credentials/list credentials/info credentials/authorize signatures/signHash

3.9.4 /auth/revoke

Revocation is forwarded to the OAuth server.

Tokens issued by the CSC implementation itself (credentials/authorize) are self contained and can not be revoked in the stateless design.

3.9.5 /oauth2/authorize

This endpoint is implemented in the "authorization" web app. A standard OAuth2 authorization flow is launched.

For "service" authorization, a "csc-gateway" private web page is published for collecting the sign-me username.

For "credential" authorization, the sign-me backend authentication page is used. Be aware that behind the scenes, this step implicitly creates a signature process at the sign-me backend.

3.9.6 /oauth2/token

This endpoint is implemented in the "authorization" web app.

It is implicitly used in the OAuth code flow process and can be explicitly used for service authorization in a sealing process.

3.9.7 /credentials/list

All usable tokens available for the service users are listed.

As "on-demand" certificates are not yet usable in the csc specification, all such tokens are filtered.

Tokens that do not provide algorithms supported by the Adobe Sign implementation are filtered.

3.9.8 /credentials/info

The sign-me token attributes are mapped to csc credential info attributes.

This process is straightforward for simple fields like "algorithm" or "length".

For other fields the following rules apply:

- As tokens that cannot be used are filtered, "status" is always "enabled".
- "authMode" is always "oauth2code"
- SCAL is always SCAL2 to enforce "hash" propagation upon authorization

3.9.8.1 CredentialInfo from sign-me Token

description	From sign-me token.tokenTypeDescription
key/status	enabled
key/algo	ecdsaWithSHA256 or RSASSA_PSS
key/len	Length of key
key/curve	ECC curve identifier if applicable
cert/status	-
cert/certificates	cert chain
cert/issuerDN	
cert/serialNumber	
cert/subjectDN	

cert/validFrom	
cert/validTo	
authMode	oauth2code or explicit when using a seal
SCAL	SCAL2
PIN/*	Available when using a seal
OTP/*	-
Multisign	Configurable in "backend.maxSignatures", Default is 255
lang	

3.9.9 /credentials/authorize

This method can be used in conjunction with seal tokens.

A credential that maps to a sign-me seal is statically configured in Adobe Sign. The credential info will show a credential with explicit PIN that can be used with credentials/authorize.

The authorization will internally use the ECDSA challenge signing protocol for sign-me seals (see "Design – Sealing").

3.9.10 /signatures/signHash

This implementation only validates and collects the signature result within this method. The "impedance mismatch" between csc and sign-me does not allow for another mapping.

3.10 Short time certificates

To satisfy regulatory requirements, after a video identification process qualified certificates have a lifetime of a few hours only. The certificate itself is created "lazy" with the start of a signature process if not yet available.

This is not directly compatible with the csc signature protocol – here we need the certificate **before** the signature process is launched, the signature process itself only receives a hash and returns the PKCS#1 structure.

To fit the two protocols, the following assumptions and decisions are made. **This design is no longer valid when one-time certificates are required!**

From sign-me side

- A "createSignature" process does create the required certificate "lazy" if the required identification level is active.
- The signature process created is valid for at least 24 hours.
- The certificate created is valid for at least 24 hours.
- The signature process can be updated with the hashes after creation.
- It is safe to create a signature process to enforce certificate creation. If it is not used, the backend system will safely discard it, alternatively it can be canceled.
- It is safe to have multiple signature processes active at the same time for the same token.
- **Crucial: The certificates created are not one-time certificates. This would put additional restrictions on the csc protocol flow otherwise.**

From csc client side

- The maximum time frame between a "credentials/info" and a corresponding use of the certificate for "credentials/authorize" and subsequent "signatures/signHash" is 1 hour

With these restrictions in place, the version starting with 2.2.x will support short term certificates by enforcing a dummy signature process, that in turn will trigger certificate creation.

3.11 Enrollment

sign-me supports self-service enrollment procedures, providing processes and user interfaces for registration and verification.

These processes are integrated in the gateway implementation.

Whenever a user is used for service authentication that is **unknown to the sign-me backend**, a registration process is created **within** the current service authentication flow and the user is forwarded to the corresponding sign-me UI. The sign-me registration process is instrumented to call back into the current service authentication.

Whenever a user is used for service authentication that is not yet **verified for using a certain token**, a verification process is created **within** the current authentication flow and the user is forwarded to the corresponding sign-me UI. The sign-me verification process is instrumented to call back into the current service authentication. As the token will not be selected until **after** service authentication, the token with the "hardest" constraints (token capabilities) is assumed to be used.

After successful service authentication, the user can be assumed to be both registered and verified for using the signature at the configured level.

Up to version 2.3.5, the registration & verification processes are instrumented with "full state" redirect URLs, i.e. upon redirect from sign-me the service authorization state was fully resurrected from the query parameters. This made it impossible to re-use a pending sign-me process as the redirect URL would have addressed stale state when a new CSC authorization has started.

With version 2.3.6 reduced state information is attached to the redirect URLs with the following implications:

- The CSC gateway needs to hold the CSC authorization in a stateful manner. The registry has a 60 minute timeout by default (see property "process.persistence.timeout").
- When returning from sign-me, the first CSC state that carries the same designator (the username to be exact) will be selected for resume.

4. Configuration

4.1 Overview

csc gateway is highly configurable and comes with a bunch of possibilities where to tweak the installation. Here's an overview of the configuration mechanics.

The backend is based on Spring infrastructure and as such you have all well-known Spring customization tools at hand.

Whenever we reference an XML based configuration file, we use the term "bean definition", when we talk about Spring properties (key/value pair definitions) we use the term "property definition".

These terms are augmented with "built-in" when we mean hardcoded, pre-deployed definitions and "custom" when we talk of individual definitions invented by your configuration.

4.1.1 Configuration location

The configuration tries to harmonize Windows and Linux based installations. We only use "abstract" location names by default. Here's the list of locations supported, where "appfamily" represents either "is-csc" or "is-auth" and "appname" either "sign-me-csc" or "sign-me-auth" for the two web applications.

Variable	Description
<appfamily>.config.name	Name of configuration file to be used without extension. Defaults to the <appname>
<appfamily>.config.user	User individual configuration data.

	Here you can store e.g. individual property files. This location has highest precedence.
<appfamily>.config.shared	System wide configuration data. Here you can store e.g. system wide property files. This location overrides the built-in configuration and is overridden by the user individual configuration
<appfamily>.data.user	User individual state data. Here is where user specific databases, caches etc. will reside.
<appfamily>.data.shared	System wide state data. Here is where system specific databases, caches etc. will reside.
<appfamily>.temp.dir	Location for temporary data, defaults to java.io.tmpdir
<appfamily>.log.dir	Location for writing log files.
<appfamily>.log.name	The name of the log file without extension Defaults to the <appname>

These variables are mapped differently on different platforms to adhere to platform specific conventions.

All of these basic variables can be overridden the standard Spring way:

- Use a command line parameter to the Java VM
-D <appfamily>.data.shared=/srv/data/foo
- Use an environment variable
<APPFAMILY>_DATA_SHARED=/srv/data/foo
- Use a key/value pair entry in a properties definition file (like "sign-me-csc.properties". It's understood that you cannot set the "<appfamily>.config.*" properties in a property file (it would have only strange effects).

If a configured directory does not exist, the web application will try to create it. For this it will need write access to the parent directory. Alternatively, you can create the directories yourself and make sure the web application has permission to write to them.

4.1.1.1 Windows

Variable	Description
<appfamily>.config.name	<appname>

<appfamily>.config.user	%USERPROFILE%/<appfamily>/config
<appfamily>.config.shared	%ProgramData%/<appfamily>/config
<appfamily>.data.user	%USERPROFILE%/<appfamily>/data
<appfamily>.data.shared	%ProgramData%/<appfamily>/data
<appfamily>.temp.dir	%AppData%/local/temp
<appfamily>.log.dir	%ProgramData%/<appfamily>/log

4.1.1.2 Linux

Variable	Description
<appfamily>.config.name	<appname>
<appfamily>.config.user	<user home>/<appfamily>/config
<appfamily>.config.shared	/etc/<appfamily>
<appfamily>.data.user	<user home>/<appfamily>/data
<appfamily>.data.shared	/var/lib/<appfamily>
<appfamily>.temp.dir	/tmp
<appfamily>.log.dir	/var/log/<appfamily>

If you are using this default, you must create the directories and grant access to the user running the servlet container.

Example: Linux hosting Tomcat servlet container

```
sudo mkdir /var/lib/<appfamily>
sudo chgrp tomcat /var/lib/<appfamily>
sudo chmod 770 /var/lib/<appfamily>

sudo mkdir /var/log/<appfamily>
sudo chgrp tomcat /var/log/<appfamily>
sudo chmod 770 /var/log/<appfamily>
```

4.1.2 Built-in configuration

csc gateway is bootstrapped with built-in definitions that are contained in the WAR deployment.

4.1.3 Custom property definition

By default, properties are defined with increasing priority from

- Built-in
classpath:builtin.properties
- Deployment level
classpath:\${<appfamily>.config.name}.properties
- System level definition
\${<appfamily>.config.shared}/\${<appfamily>.config.name}.properties
- User level definition
\${<appfamily>.config.user}/\${<appfamily>.config.name}.properties

4.1.4 Custom bean definition

Custom bean definitions are not supported with this implementation.

4.1.5 Using profiles

You can use Spring profiles to parameterize your configuration. A profile allows to bundle beans and properties and give them a meaningful name.

When starting the server, you can activate any of these profiles by defining

```
spring.profiles.active=<comma separated profile names>
```

This definition can be made in any of the well-known ways, either

- as environment variable
- as system property
- in your \${<appfamily>.config.name}.properties

To restrict a bean definition to a specific profile, you can add a section in your bean definition file like this

```
...
<beans profile="profilename">
...
</beans>
...
```

Any definition contained in the "<beans>" element will be used only if the corresponding profile is activated.

In addition, the application will read specific property files in addition to "\${<appfamily>.config.name}.properties" whose name match "\${<appfamily>.config.name}-<profilename>.properties".

The priority (increasing to the bottom) is

- System level definition
 - \${<appfamily>.config.shared}/\${<appfamily>.config.name}.properties

- `${<appfamily>.config.shared}/${<appfamily>.config.name}-<profilename>.properties}`,
In the order of profile definition, first one highest
- User level definition
 - `${<appfamily>.config.user}/${<appfamily>.config.name}.properties`
 - `${<appfamily>.config.user}/${<appfamily>.config.name}-<profilename>.properties}`,
In the order of profile definition, first one highest

4.1.6 String expansion syntax

spring will automatically expand strings in the form "...\${foo}..." from the pool of available properties. If a property is missing, configuration fails.

In addition, we have components that execute string expansion at runtime (e.g. the URL query string suffixes) – these components use "\${..}" syntax, too. If you need to configure a property that gets expanded at runtime, you must prevent the static spring expansion that takes place at application startup.

You must either use a spring expression language construct like

```
#('$' + '{username}')
```

or a application specific shortcut

```
?{username}
```

that will get automatically replaced internally.

4.2 authorization app properties

jws.hmacKey	
String	The key that is used for HMAC message authentication Default jws.hmacKey=123456
jws.tokenValidity	
int	The validity period in seconds for a token issued by the OAuth2 service. Default (10 min) jws.tokenValidity =600
authRegistry.retentionTime	

int	<p>The maximum time in milliseconds an authorization request is retained in the OAuth2 server context. This is the maximum duration a consent gathering process may take before the request itself is deleted from memory.</p> <p>Default (10 min)</p> <p>authRegistry.retentionTime =600000</p>
backend.cscServer	
URL	<p>The authorization app is associated with a csc sign-me backend and needs private access to internal data and features. The csc services are accessed as <code>\${backend.cscServer}/csc/v0/*</code>.</p> <p>The csc server name needs to be defined here if it is not deployed within the same container as the authorization web application itself.</p> <p>Default</p> <p>backend.cscServer=<current server></p> <p>Example</p> <p>backend.cscServer=http://localhost:8080</p>
templateEngine.templates	
string	<p>A spring resource denoting a directory where the web UI templates are looked up.</p> <p>Default</p> <p>file:\${is-auth.config.shared}/templates/</p>
templateEngine.prefix	
string	<p>A spring resource denoting a directory where the web static resources are looked up.</p> <p>Default</p> <p>file:\${is-auth.config.shared}/static/</p>
consentController.alertFor	
string	<p>A list of error codes that should be intercepted when returned from csc identification.</p> <p>Default</p> <p>SignerNotExisting,SignerStatusInsufficient</p>

4.3 csc app properties

jws.hmacKey	
String	<p>The key that is used for HMAC message authentication</p> <p>Default</p> <p>jws.hmacKey=Simple default HMAC key for SAD signing</p>
jws.tokenValidity	

int	The validity period in seconds for an SAD issued by credentials/authorize. Default (10 min) jws.tokenValidity =600
backend.oauth2ClientId	
String	The OAuth2 client id to perform a password grant. This is used whenever a csc client performs a direct auth/login to delegate the authentication to the OAuth2 server.
backend.oauth2ClientSecret	
String	The OAuth2 client password to perform a password grant. This is used whenever a csc client performs a direct auth/login to delegate the authentication to the OAuth2 server
backend.maxSignatures	
int	The maximum number of signatures that can be applied with a single user consent
backend.keyValueStore	
String	The path where the key/value store artifacts are persisted Default: \${is-csc.data.shared }/key-value-store

service.v0.info.specs	
String	The “specs” property of the info response Default is 0.1.7.9
service.v0.info.name	
String	The “name” property of the info response
service.v0.info.description	
String	The “description” property of the info response
service.v0.info.logo	
String	The “logo” property of the info response
service.v0.info.region	
String	The “region” property of the info response
service.v0.info.authTypes	
String	The “authTypes” property of the info response
service.v0.info.methods	
String	The “methods” property of the info response
service.v0.info.oauth2Server	
String	The “oauth2Server” property of the info response

service.v1.info.specs	
String	The “specs” property of the info response Default is 1.0.3.0
service.v1.info.name	
String	The “name” property of the info response
service.v1.info.description	
String	The “description” property of the info response
service.v1.info.logo	
String	The “logo” property of the info response
service.v1.info.region	
String	The “region” property of the info response
service.v1.info.authTypes	
String	The “authTypes” property of the info response
service.v1.info.methods	
String	The “methods” property of the info response
service.v1.info.oauth2Server	
String	The “oauth2Server” property of the info response

sign-me.endpointAddress	
URL	<p>Defines the sign-me endpoint as disclosed by D-Trust.</p> <p>This is only the fallback for a missing "default" account in the sign-me account registration.</p> <p>Example</p> <p>sign-me.endpointAddress=https://cloud-ref-sp.sign-me.de:443/api/v2</p>
sign-me.endpointUser	
String	<p>Defines the user for performing HTTP basic authentication against the sign-me endpoint. This value is disclosed by D-Trust.</p> <p>This is only the fallback for a missing "default" account in the sign-me account registration.</p> <p>Example</p> <p>sign-me.endpointUser=<endpoint username></p>
sign-me.endpointPassword	
String	<p>Defines the password for performing HTTP basic authentication against the sign-me endpoint. This value is disclosed by D-Trust.</p> <p>This is only the fallback for a missing "default" account in the sign-me account registration.</p> <p>Example</p> <p>sign-me.endpointPassword=<endpoint password></p>
sign-me.partnerUser	
String	<p>Defines your partner username to be used for sign-me. This value is disclosed by D-Trust.</p> <p>This is only the fallback for a missing "default" account in the sign-me account registration.</p> <p>Example</p> <p>sign-me.partnerUser =<partner user></p>
sign-me.partnerPassword	
String	<p>Defines your partner password to be used for sign-me. This value is disclosed by D-Trust.</p> <p>This is only the fallback for a missing "default" account in the sign-me account registration.</p> <p>Example</p> <p>sign-me.partnerPassword =<partner password></p>
sign-me.partnerRole	
String	<p>Defines your partner role to be used for sign-me. This value is disclosed by D-Trust.</p> <p>This is only the fallback for a missing "default" account in the sign-me account</p>

	<p>registration.</p> <p>Example</p> <p>sign-me.partnerRole=<partner role></p>
sign-me.tokenCapabilities	
String	<p>A list of token capabilities that are used when requesting the available tokens for the user.</p> <p>Default</p> <p>SEALING_ADVANCED,SEALING_QUALIFIED,SIGNATURE_ADVANCED,SIGNATURE_QUALIFIED</p> <p>Example</p> <p>sign-me.tokenCapabilities=SIGNATURE_QUALIFIED</p>
sign-me.signature.queryString	
String	<p>This string will be appended to the sign-me signature URL upon open.</p> <p>For the syntax see the explanation in chapter 4.1.6</p> <p>Example:</p> <p>"LO=M&USERNAME=?{username}"</p> <p>Default</p> <p>"LO=M&USERNAME=?{username}"</p>
sign-me.verification.type	
String	<p>One of the sign-me well known identity verification types, e.g.</p> <ul style="list-style-type: none"> • VIDEO_4EYE • WEB_FT_IDENT_4EYE • POS_FT_IDENT_4EYE • E_ID • FT_E_ID <p>This string is intentionally not constrained to any enumeration to allow for changes in the backend and the usage of non-public features.</p> <p>Default: FT_E_ID</p>
sign-me.verification.queryString	
String	<p>This string will be appended to the sign-me verification URL upon open</p> <p>For the syntax see the explanation in chapter 4.1.6</p> <p>Example:</p> <p>"LO=M&USERNAME=\${username}"</p> <p>Default</p> <p>"LO=M&USERNAME=\${username}"</p>
sign-me.registration.queryString	

String	<p>This string will be appended to the sign-me registration URL upon open</p> <p>For the syntax see the explanation in chapter 4.1.6</p> <p>Example:</p> <p>"LO=M"</p> <p>Default</p> <p>"LO=M&EMRO=true"</p>
sign-me.registration.checkConfirmationEmail	
boolean	<p>Flag if email confirmation will be checked in the enrollment process</p> <p>Default: false</p>
sign-me.seal.authenticationTokenFolder	
String	<p>A folder name that contains all token files for seal authentication</p> <p>Default: \${is-cec.config.shared}/authenticationTokens</p>
sign-me.seal.authenticationMaxRetries	
Int	<p>The maximum number of retries for entering the correct authentication token pin. After exceeding this value, the account token is permanently locked.</p> <p>You must manually remove or update the key/value store with the authentication token meta-data at</p> <p>\${backend.keyValueStore}/auth-<tokenname>.json</p> <p>Default: 3</p>
process.persistence.timeout	
long	<p>The time in milliseconds before an authorization context is wiped from the registry.</p> <p>Default: 3600000 (60 minutes)</p>

4.4 csc app deprecated properties

Earlier versions only supported v.0.1 of the spec and as such the following properties were unique – this is no longer the case.

Processing is now extended so that these properties are the “backup defaults” that are overridden with the service version specific values **if they are present**.

backend.name	
String	The “name” property of the info response
backend.description	
String	The “description” property of the info response

backend.logo	
String	The "logo" property of the info response
backend.region	
String	The "region" property of the info response
backend.authTypes	
String	The "authTypes" property of the info response
backend.methods	
String	The "methods" property of the info response
backend.oauth2Server	
String	The "oauth2Server" property of the info response

4.5 Registering sign-me accounts

To support accounting (based on the adobe account_token), multiple sign-me accounts can be registered.

To register an account, you have to provide the account configuration in the "accounts.json" file in the "is-csc" configuration folder.

For each valid account the following information must be provided:

accountId	
String	The account id from the adobe registry.
clientId	
String	An optional clientId to restrict the application of this account to a specific OAuth2 client
partnerUser	
String	Defines your partner username to be used for sign-me. This value is disclosed by D-Trust. Example partnerUser =<partner user>
partnerPassword	
String	Defines your partner password to be used for sign-me. This value is disclosed by D-Trust. Example partnerPassword =<partner password>
partnerRole	
String	Defines your partner role to be used for sign-me. This value is disclosed by D-Trust.

	Example partnerRole=<partner role>
endpointAddress	
String	Defines the sign-me endpoint as disclosed by D-Trust. Example endpointAddress=https://cloud-ref-sp.sign-me.de:443/api/v2
endpointUser	
String	Defines the user for performing HTTP basic authentication against the sign-me endpoint. This value is disclosed by D-Trust. Example endpointUser=<endpoint username>
endpointPassword	
String	Defines the password for performing HTTP basic authentication against the sign-me endpoint. This value is disclosed by D-Trust. Example endpointPassword=<endpoint password>
tokenCapabilities	
String	You can define the filtered token capabilities for an individual account. This property overrides the global "sign-me.tokenCapabilities" property for this specific account. Example tokenCapabilities=<comma separated capability list>
verificationType	
String	You can define the identity verification type for an individual account. This property overrides the global "sign-me.verification.type" property for this specific account. Example verificationType=FT_EID
authenticationTokens	
Array of AuthenticationToken	An array of all authentication tokens that can be used for sealing with this account.

The AuthenticationToken has the following properties

Id	
String	The id of the authentication token.

	This must match the credential id that is configured with Adobe Sign.
fileName	
String	The name of the authentication token P12 file in the respective folder. If omitted, this computes to <id>.p12
orgName	
String	The organization name for the authentication token. If omitted, this defaults to the prefix (up to “_”) of the id
userName	
String	The user that is associated with the sealing token in the sign.me backend. This entry is required.

Example

```
[
{
  "accountId": "default",
  "partnerUser": "<partner user default>",
  "partnerPassword": "<partner user default>",
  "partnerRole": "PARTNER",
  "endpointAddress": "<endpoint address default>",
  "endpointUser": "<endpoint user default>",
  "endpointPassword": "<endpoint user default>",
  "tokenCapabilities": "SIGNATURE_QUALIFIED"
},
{
  "accountId": "<adobe account a>",
  "partnerUser": "<partner user a>",
  "partnerPassword": "<partner user a>",
  "partnerRole": "PARTNER",
  "endpointAddress": "<endpoint address a>",
  "endpointUser": "<endpoint user a>",
  "endpointPassword": "<endpoint user a>",
  "tokenCapabilities": "SIGNATURE_ADVANCED"
},
{
  "accountId": "<adobe account b>",
  "clientId": "clientAdvanced",
  "partnerUser": "<partner user b>",
  "partnerPassword": "<partner user b>",
  "partnerRole": "PARTNER",
  "endpointAddress": "<endpoint address b>",
  "endpointUser": "<endpoint user b>",
  "endpointPassword": "<endpoint user b>",
  "tokenCapabilities": "SIGNATURE_ADVANCED"
},
{
  "accountId": "<adobe account b>",
  "clientId": "clientQualified",
  "partnerUser": "<partner user b>",
  "partnerPassword": "<partner user b>",
  "partnerRole": "PARTNER",
  "endpointAddress": "<endpoint address b>",
  "endpointUser": "<endpoint user b>",
  "endpointPassword": "<endpoint user b>",
  "tokenCapabilities": "SIGNATURE_QUALIFIED"
},
{
  "clientId": "cid-3",
  "partnerUser": "partnerUser-tokens",
  "partnerPassword": "partnerPassword-tokens",
  "partnerRole": "partnerRole-tokens",
  "endpointAddress": "endpointAddress-tokens",
  "endpointUser": "endpointUser-tokens",
  "endpointPassword": "endpointPassword-tokens",
  "authenticationTokens": [
    {
      "id": "token1", // required, configured with Adobe seal administration
      "fileName": "org_token1.p12", // optional, derived from id
      "orgName": "org", // optional, derived from fileName
      "userName": "user1" // required
    }
  ]
}
]
```

```

    },
    {
      "id": "token2",
      "fileName": "org_token2.p12",
      "orgName": "org",
      "userName": "user2"
    }
  ]
}
]

```

If no "default" account is defined, a "default" account is created using the corresponding properties from the csc app. If these properties are not available, a default account is not created. In this case the signature request will fail if either an account_token is not sent or the corresponding account is not configured.

4.6 Registering OAuth2 clients

The OAuth2 service gives access to its API to registered clients only. This is comparable to a standard client registration with "google OAuth" services or a Keycloak service.

For using the application with "Adobe Sign", you have to enter the IDs and credentials that have been negotiated with the Adobe representatives here. You can find an example in the "examples" folder of the distribution.

To register a client, you have to provide the client configuration in the "clients.json" file.

For each valid client the following information must be provided:

id	
String	<p>The OAuth2 client id.</p> <p>This identifies a valid OAuth2 client. The OAuth client uses this ID to identify himself when requesting an authentication.</p>
secret	
String	<p>The OAuth2 client secret.</p> <p>This identifies a valid OAuth2 client. The OAuth client uses the secret to authenticate identify himself (when requesting a token).</p> <p>This is sensitive information and should not be stored in plaintext. The implementation supports the Spring "password encoding" scheme, ensuring that only a cryptographically hashed version of the password needs to be stored.</p> <p>An encoded password has for example the form</p> <pre>{bcrypt}\$2a\$10\$zuQl84DPiJLkcRWfNlzc.yrHHb/4Ylu3I.XU6uH5.78moc6oR2p2</pre> <p>This representation can be created using spring primitives directly or the <code>"/authorization/tools/encode"</code> service in the "authorization-admin" web</p>

	application. A password without an {<id>} prefix is assumed plaintext.
name	
String	An internal human readable representation
grantTypes	
Array of String	The list of OAuth grant types allowed for the client. This list members are one of the supported OAuth2 grant types <ul style="list-style-type: none"> • authorization_code • client_credentials • password
redirectUrls	
Array of Regex Pattern	The list of redirect URIs allowed for the client. The redirect used for a concrete OAuth authentication request must be one of the URI registered here.

The registry of valid clients is read from a "clients.json" file in JSON format. This file is looked up in the directories defined below

- \${ is-auth.config.user}
- \${ is-auth.config.shared}

Example

```
[
  {
    "id": "example",
    "name": "example.com Demo App",
    "secret": "you never know",
    "grantTypes": [
      "authorization_code"
    ],
    "redirectUrls": [
      "http://www\\.example\\.com/.*"
    ]
  }
]
```

Pay especially attention to the string format for "redirectUrls". First, these strings are regular expressions, so you must escape special characters like ".". Second, these are JSON string, so you must escape the escape.

4.7 Web application root

In many production environments the web application container itself cannot know what is the fully qualified external URL for the services. This is most often caused by a reverse proxy.

While most of the time relative addresses are fine, sometimes the server needs to construct fully qualified URL, for example for callback or redirect addresses.

While the server does its best to derive what may be the URL from an external point of view by examining de-facto standard HTTP headers, there may be times this is not working, e.g. when a proxy is misconfigured and cannot be changed in the scope of your installation.

In such special cases you can override the automatic detection by setting the root URL explicitly using the property

```
de.intarsys.application.rootUrl
```

These are the locations where we look up the property from lowest to highest precedence.

You can set the property in the web.xml

```
<context-param>
  <param-name>de.intarsys.application.rootUrl</param-name>
  <param-value>https://my.server.com</param-value>
</context-param>
```

You can use an environment variable

```
DE_INTARSYS_APPLICATION_ROOTURL=https://my.server.com
```

You can set the property using a Java system property

```
-Dde.intarsys.application.rootUrl=https://my.server.com
```

4.8 Logging

With regard to logging cloud suite tries to come up with a default setup that can be used out of the box.

Internally, the Logback logging framework is used. The features and syntax of Logback are beyond the scope of this documentation. There are many resources available on the internet.

4.8.1 Override logging

When starting up, Logback tries to configure itself using the default "logback.xml", situated **anywhere** in a root package on the class path (or known from the Logback command line options). All standard Logback

functionality is available. If you are happy with this and provide a configuration, you can skip the rest.

As soon as cloud suite is aware of this "external" Logback configuration, it skips all further activities. You just have to be **sure** that you do not have any of the internal context variables at your hand at this moment in time.

Only if no external configuration is applied, the following steps are performed.

4.8.2 Builtin logging

If no external configuration is applied, at the earliest moment in the application lifecycle (in a Spring listener), we perform a relaunch of the Logback environment - this time with the well-known cloud suite variables established.

We then try to load a "\${<appfamily>.config.user}/logback.xml" and "\${<appfamily>.config.shared}/logback.xml", if this fails, we fall back to an internal default Logback configuration.

This default will write all its output to the "\${<appfamily>.log.dir}" directory using the "\${<appfamily>.log.name}.log" as the log file name - a platform dependent location as stated above, using the log level "INFO".

The following Logback variables are available for your use within your logback.xml at this stage.

Logback variable	Definition
<appfamily>.config.shared	\${<appfamily>.config.shared}
<appfamily>.config.user	\${<appfamily>.config.user}
<appfamily>.data.shared	\${<appfamily>.data.shared}
<appfamily>.log.dir	\${<appfamily>.log.dir}
<appfamily>.log.name	<appname>
<appfamily>.log.level	\${<appfamily>.log.level}:INFO
config.shared	\${<appfamily>.config.shared}
config.user	\${<appfamily>.config.user}
data.shared	\${<appfamily>.data.shared}
log.dir	\${<appfamily>.log.dir}
log.name	<appname>
log.level	\${<appfamily>.log.level}:INFO

4.8.3 Log tweaks

4.8.3.1 Directory

If you simply want to set another target directory, you can just set a property like this. This will be forwarded to the built-in log definition.

```
<appfamily>.log.dir=/srv/logs
```

4.8.3.2 Name

If you simply want to use another log file name, you can just set a property like this. This will be forwarded to the built-in log definition.

```
<appfamily>.log.name=foobar
```

4.8.3.3 Level

If you simply want to set another logging level, you can just set a property like this. This will be forwarded to the built-in log definition.

```
<appfamily>.log.level=DEBUG
```

4.8.3.4 "debug" profile

If you need more detailed information for debugging, you can try add the "debug" profile. This is intended to give more overall diagnostic information.

At the moment, this will register the "LoggingFeature" for the JAXRS environment and as such will give many details on the communication protocol.

```
spring.profiles.active=debug
```

4.9 Licenses

The product may require valid licenses for execution.

In this case, licenses are obtained from intarsys. A limited demo license is included with the product for basic usage.

You can provide new licenses by copying the license files either to

```
${<appfamily>.config.shared}/licenses
```

or

```
${<appfamily>.config.user}/licenses
```

Upon startup, all licenses that have been picked up are logged to the log file.

```
[04.05.2018-10:36:18.622][I][d.i.tools.license ][localhost-startStop-1][]  
loading licenses from 'C:\ProgramData\is-csc\config'  
[04.05.2018-10:36:18.653][I][d.i.tools.license ][localhost-startStop-1][]  
license 'de.intarsys.product.is-csc.signme; 8; intarsys.de; 12/01/2017;  
06/30/2018; id=de.intarsys.product.is-csc.signme; account_automation_cli=-  
1:day; account_automation_batch=-1:day; bundle=professional; batchsize=-1;  
account_automation_api=-1:day; ' loaded
```

5. Web apps

5.1 "csc"

This web application implements the csc remote signing API as specified in [3].

5.2 "authorization"

This web application implements an OAuth2 compatible authorization server.

The services are used internally in the csc application and are not intended for other use.

5.3 "authorization-admin"

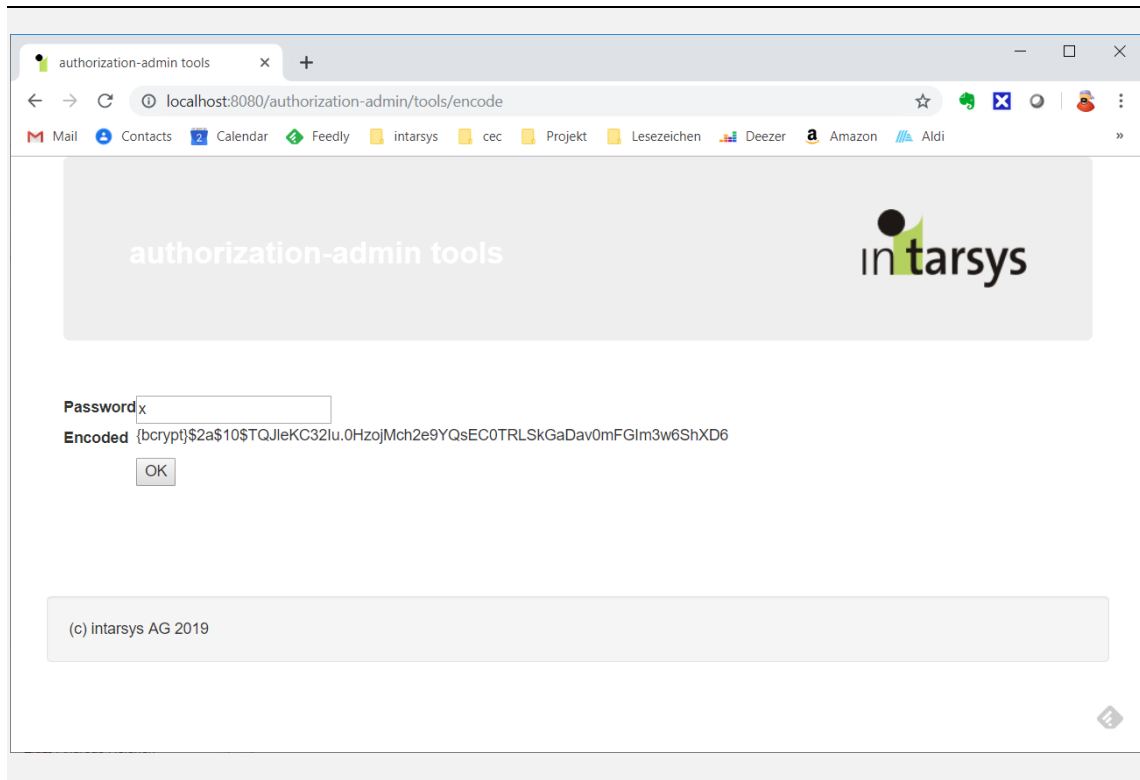
This web application provides administrative functions for the "authorization" app.

The application should not be publicly available.

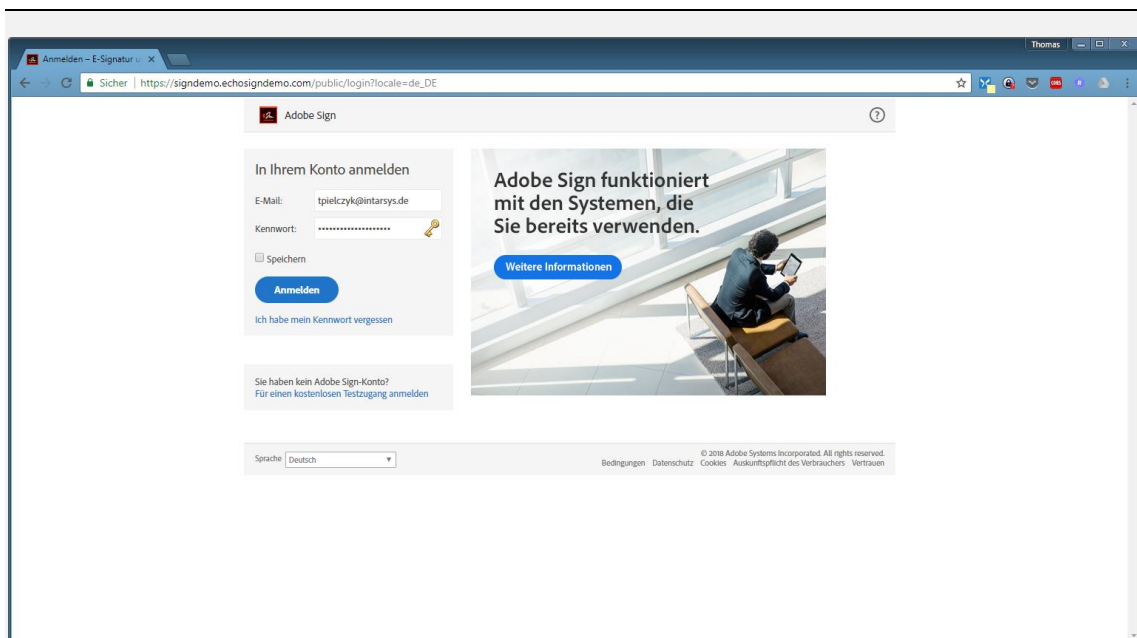
5.3.1 UI

5.3.1.1 /authorization-admin/tools/encode

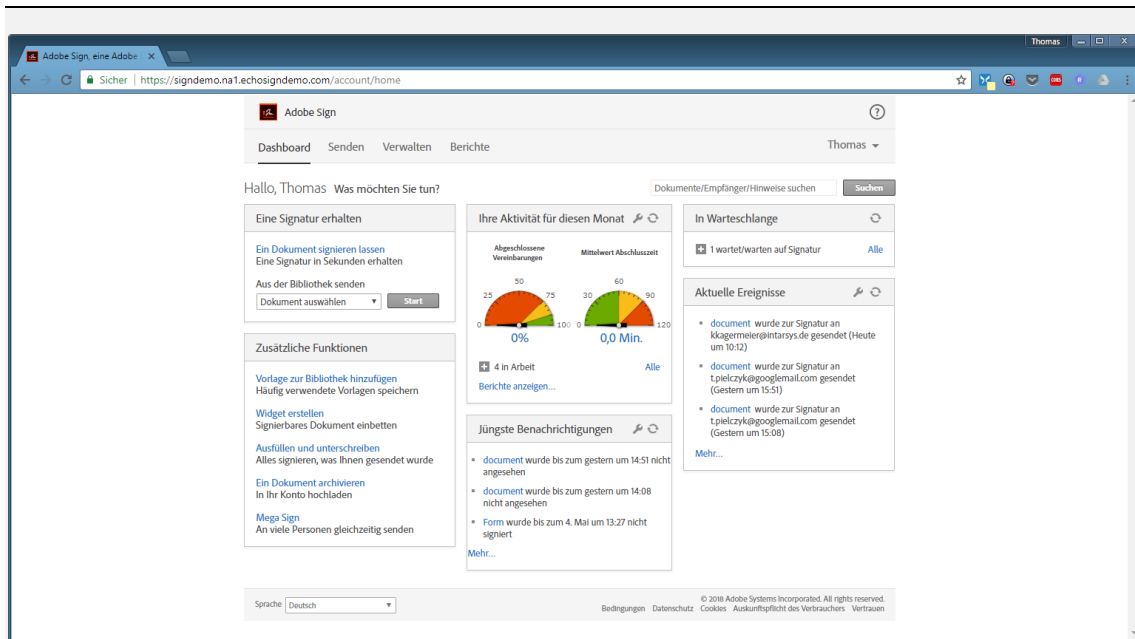
This page allows you to enter a password and will return the default hashed version, ready to be used in the configuration files.



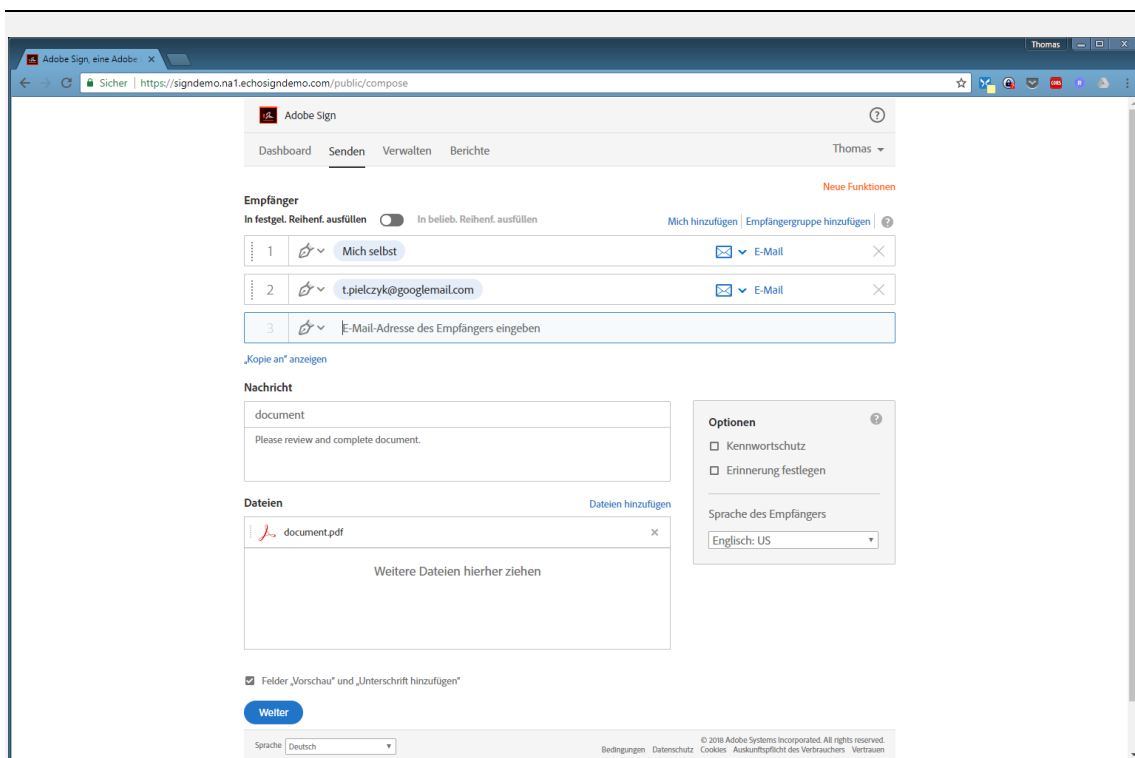
6. Adobe Sign screen flow



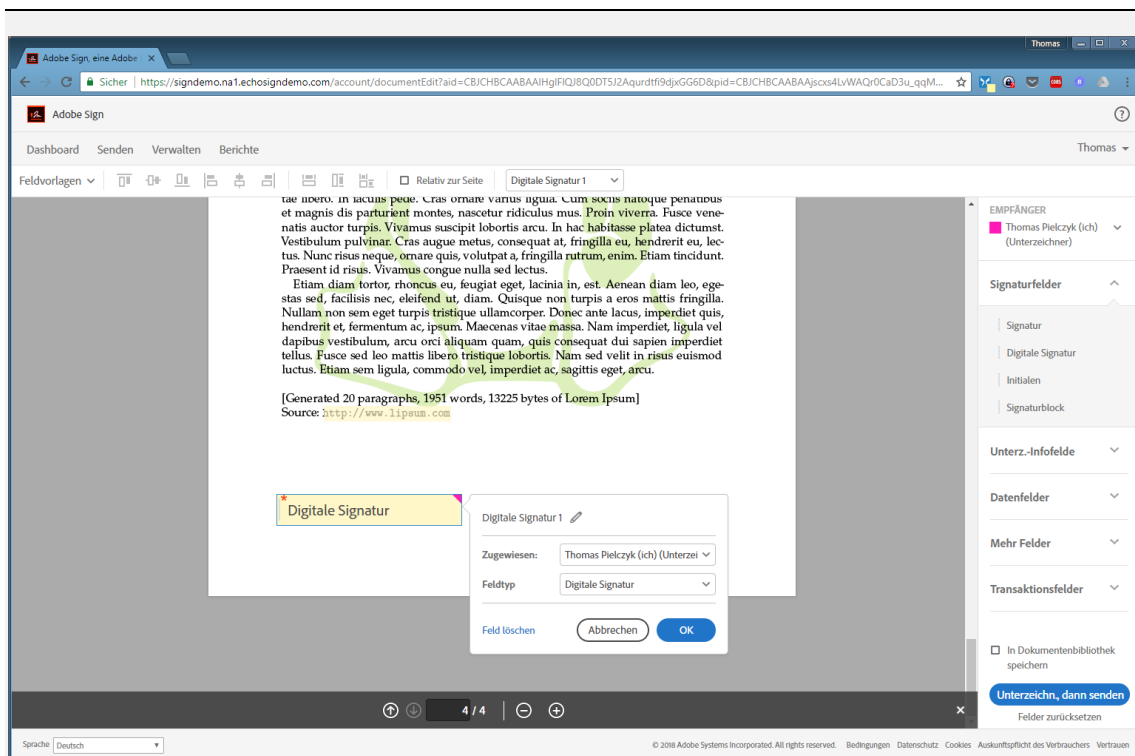
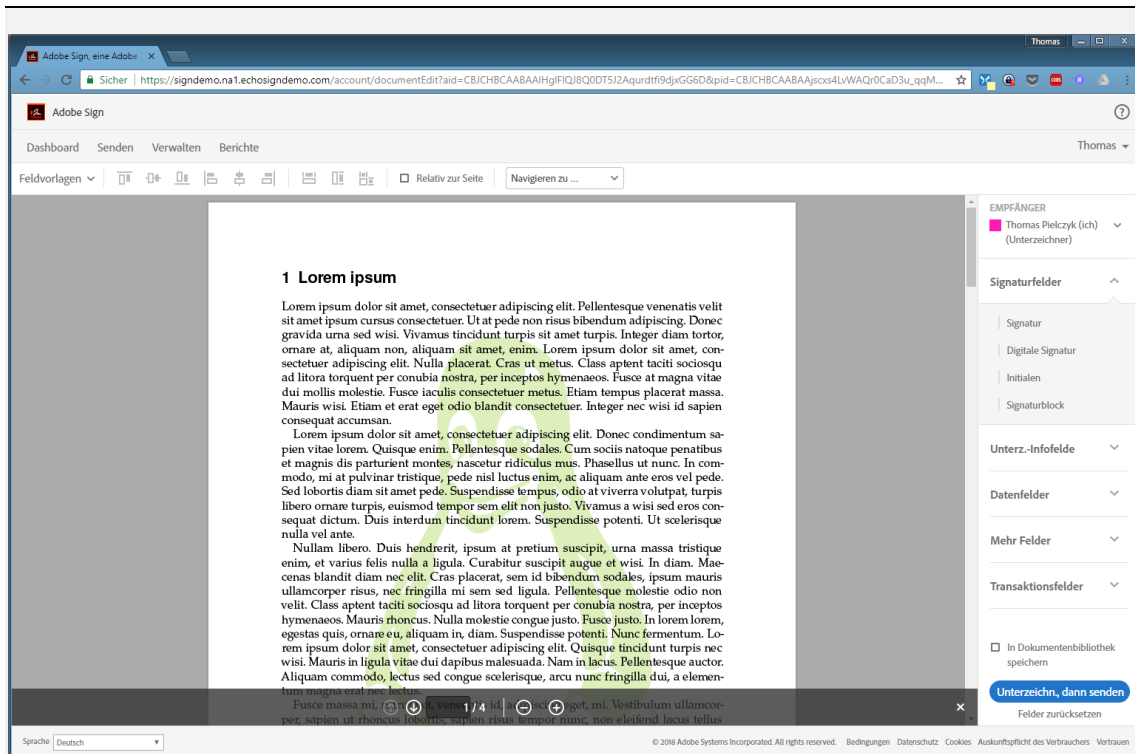
Start at
<https://signdemo.echosigndemo.com>
and logon with your credentials.

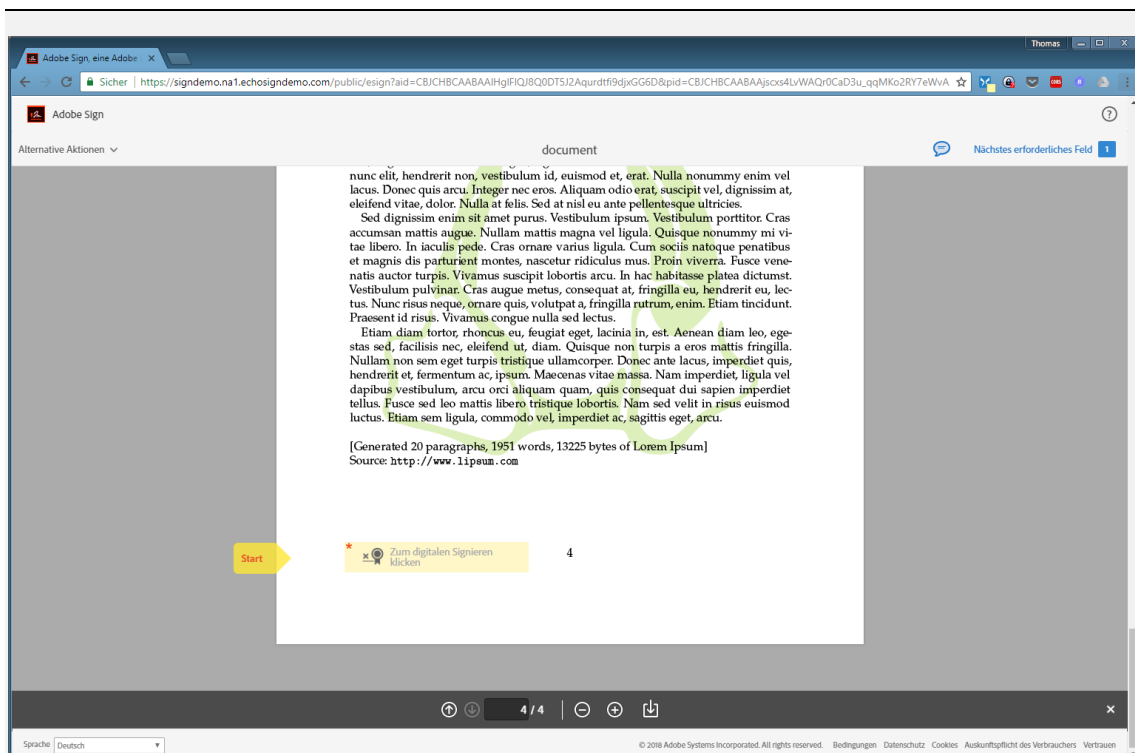
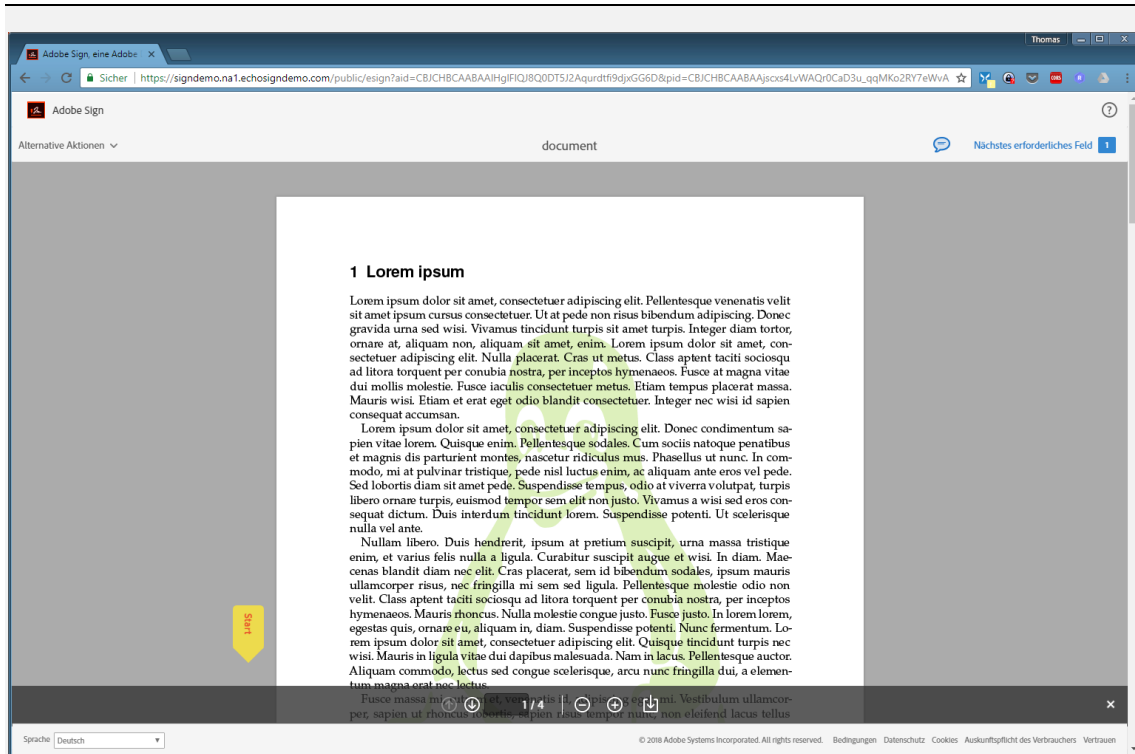


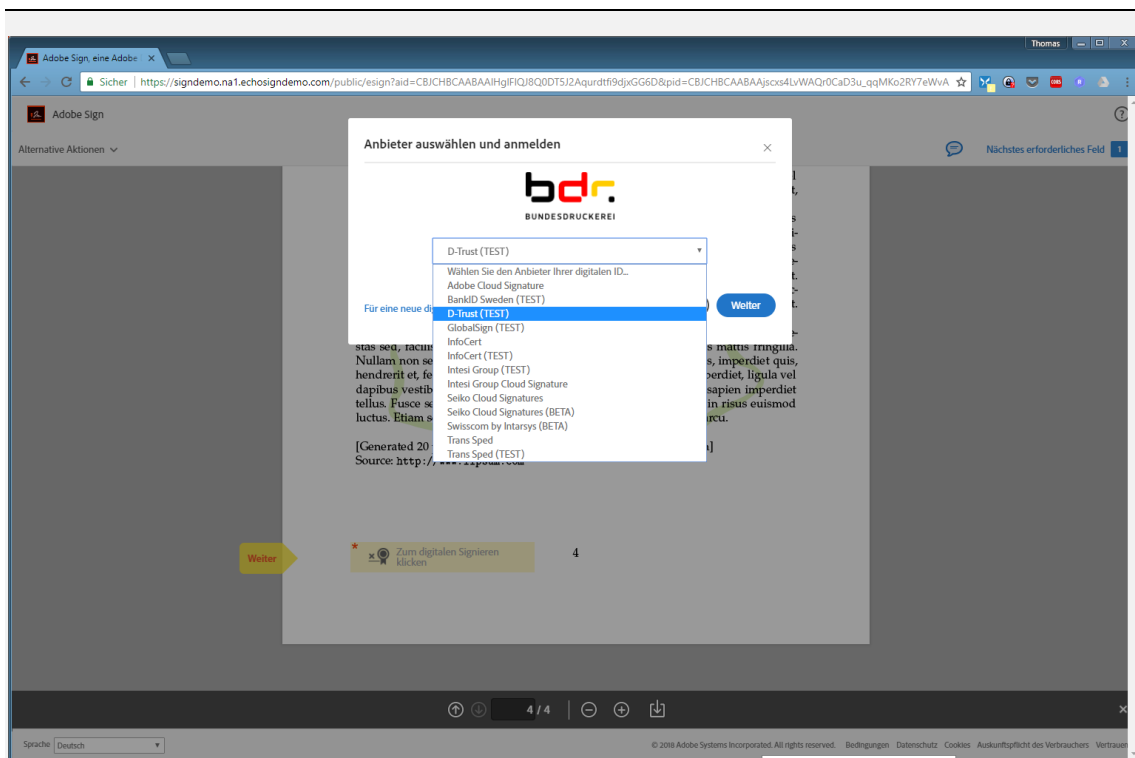
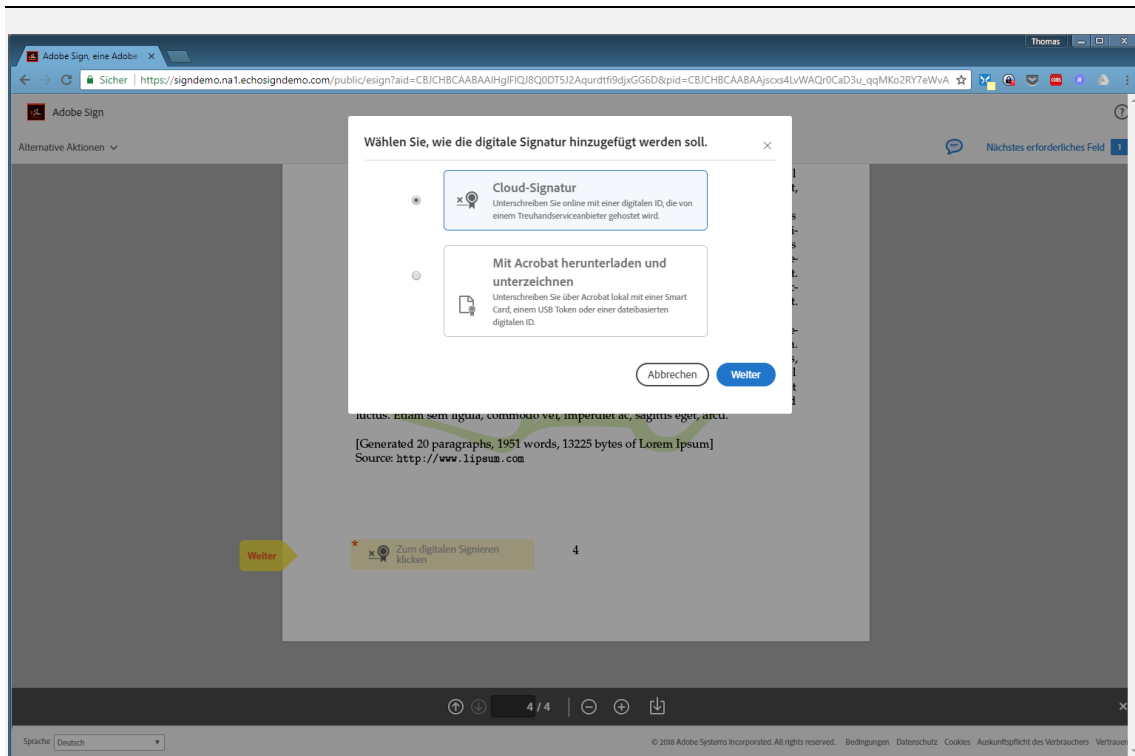
You will see the Adobe Sign dashboard.

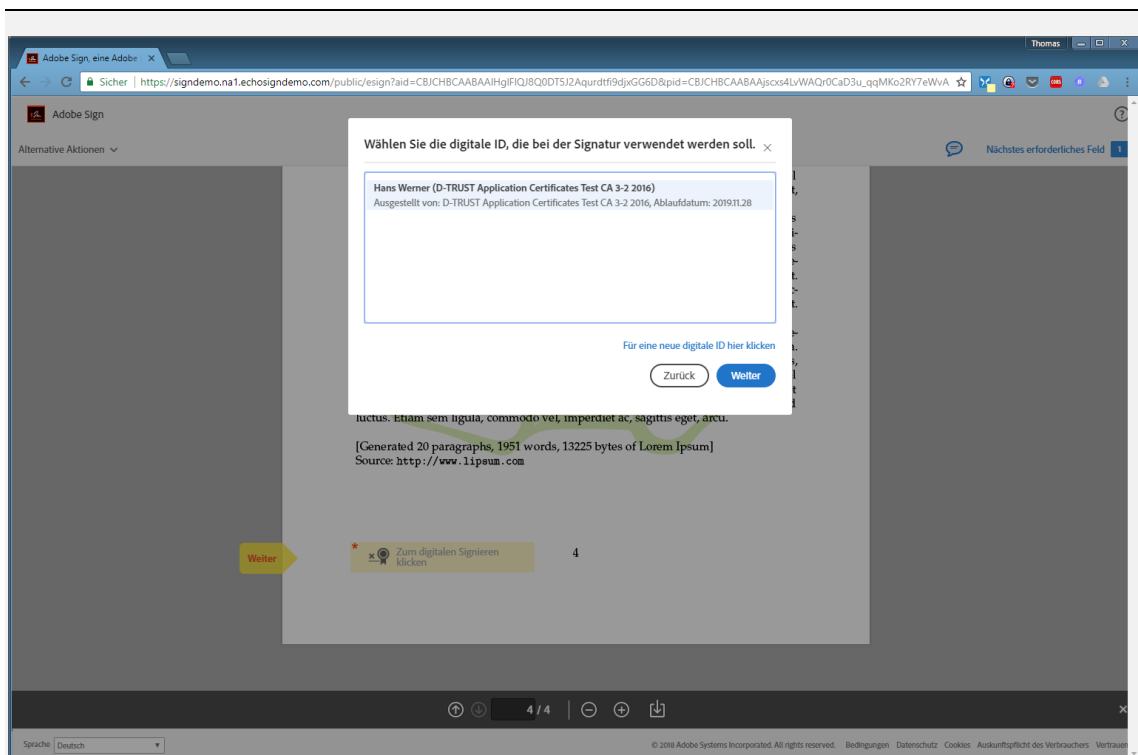
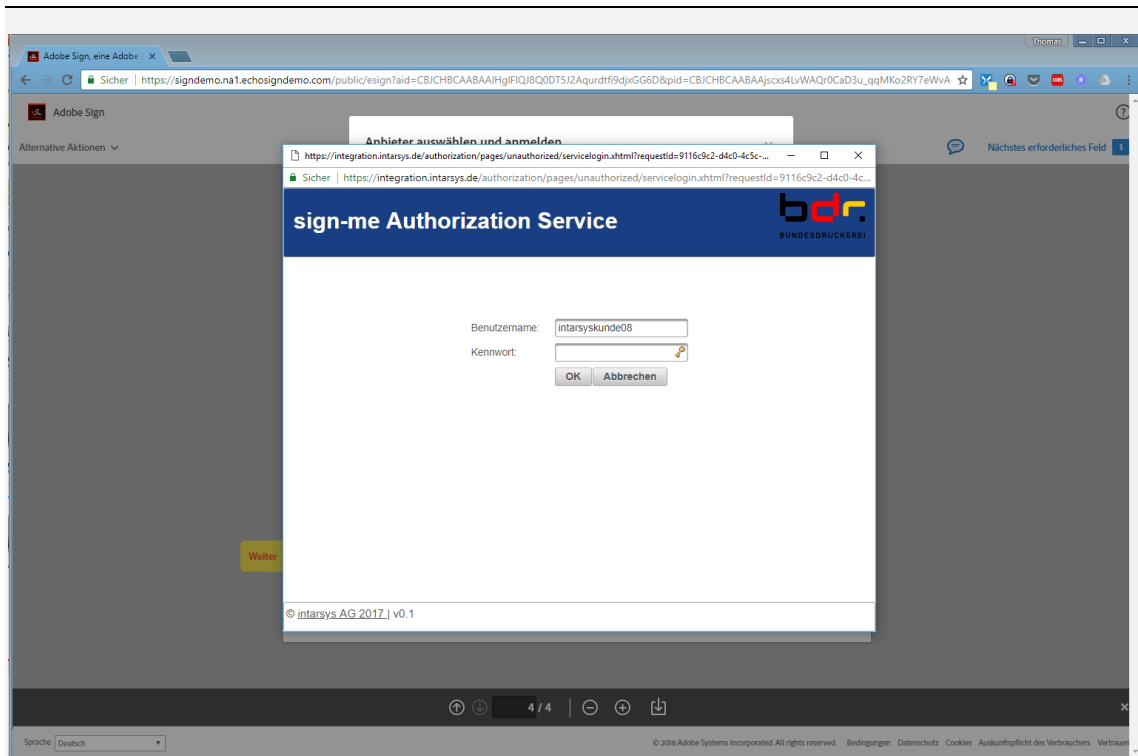


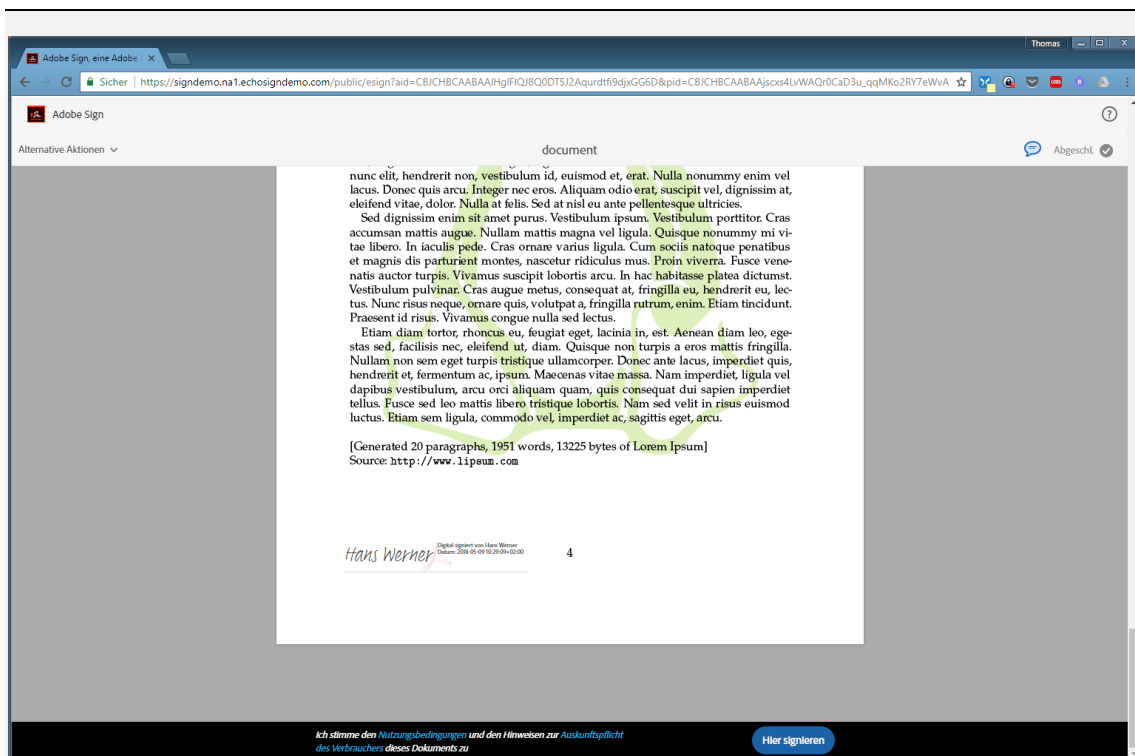
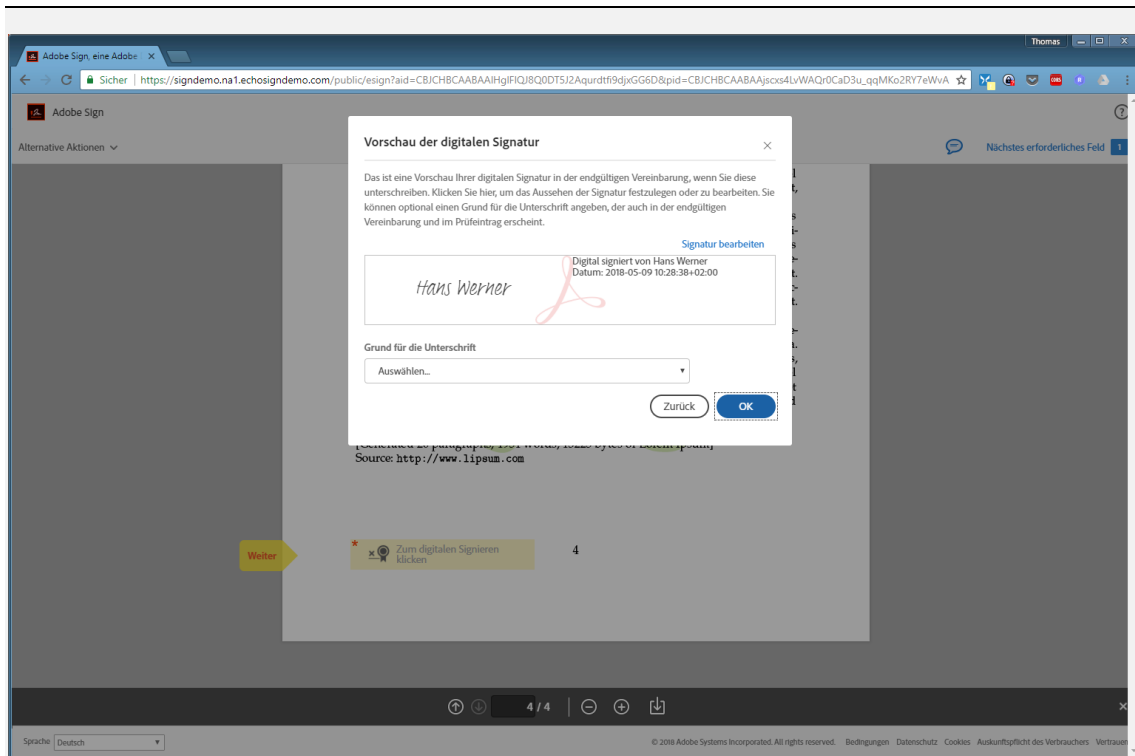
Send a document to be signed.

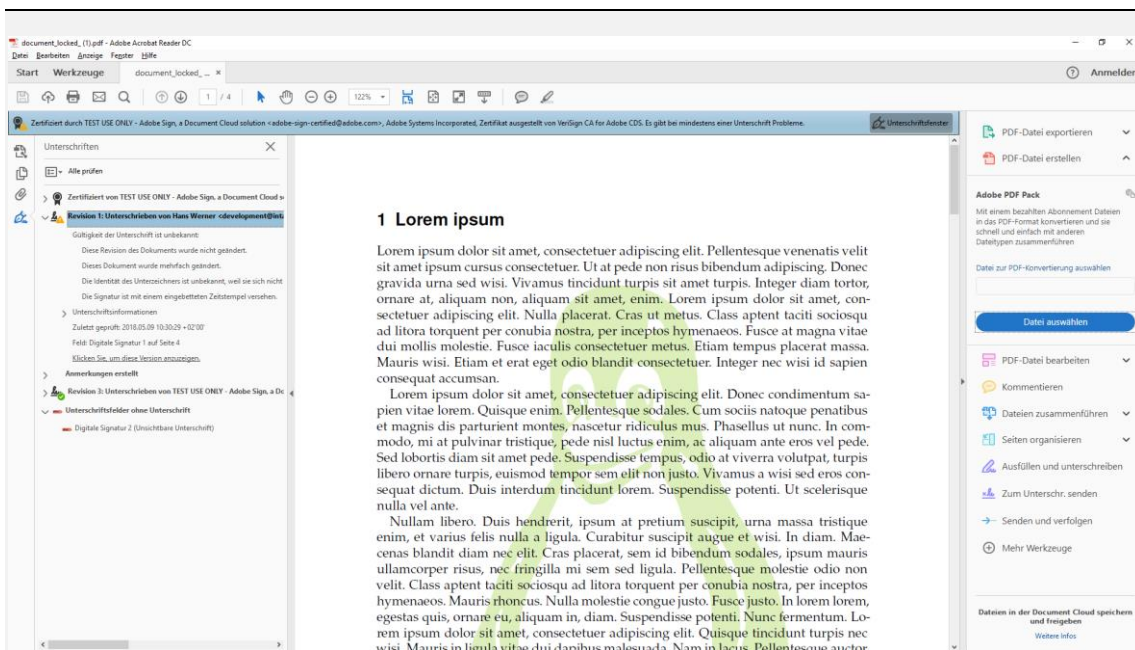
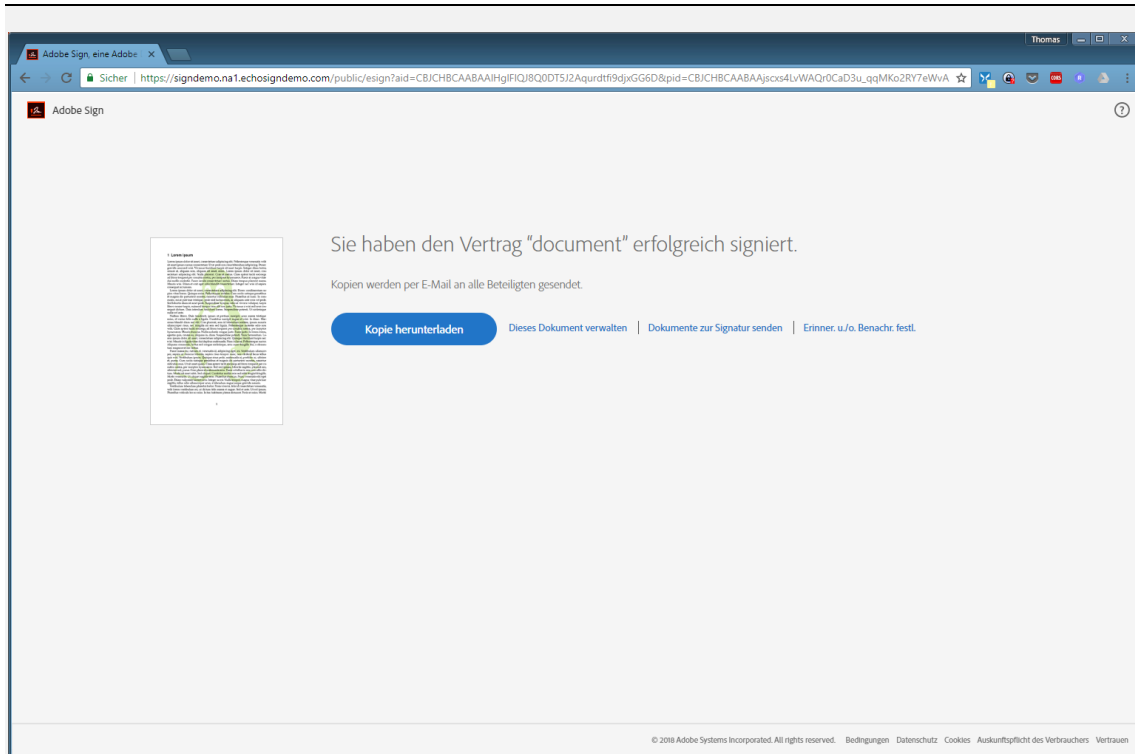












7. UI customization

7.1 Overview

The web apps may come with builtin web UIs to allow user interaction. These UIs are created using Thymeleaf, a widespread template engine and rely on Spring web MVC.

To ease customization, both templates and static resources can be overwritten for your installation.

7.2 authorization app

7.2.1 Consent pages

The authorization app publishes a consent page and the required resources. The template is named "sign-me /serviceConsent.html". To overwrite this resource, you can add a file "sign-me /serviceConsent.html" in the folder "\${is-auth.config.shared}/templates/" (or the folder configured in the property "templateEngine.templates").

If you have static resources like this

```
<link rel="stylesheet" type="text/css" th:href="@{/sign-me  
/res/css/general.css}">
```

you can put them (or overwrite the original ones) in the folder "\${is-auth.config.shared}/static/" (or the folder configured in the property "templateEngine.static").

7.2.2 Error pages

When the spring controller encounters an error, the error handling will trigger one of two templates:

In case of an expired authentication request, the "error-expired.html" view is shown.

In all other cases, the "error.html" view is shown.

The model for the error pages is

id	
String	A UUID that can be used for correlation with the log
url	
String	The request URL that caused the failure
exception	
Exception	The exception that was thrown
dateTime	
LocalDateTime	The local date/time object when the model was created

7.2.3 Hints

There is a complete example in the folder "examples/web-customization".

Be aware that you must conform to the existing form API.

Thymeleaf caches the templates – changes need a restart.

7.3 csc app

The csc app has currently no builtin web pages.

8. NLS

8.1 NLS resolving

The application relies on the Spring string expansion for NLS resolving, allowing a very flexible use of the concept.

In general, you know strings of the form

```
${code}
```

are expanded, normally against the various properties available to the Spring runtime.

NLS support introduces a dedicated resolver that is applied to all property expressions that start with a "nlsmsg." The complete syntax is

```
nlsmsg.<package>.<resource>#<code>
```

where "package" is a path name according to the Java package naming convention (e.g. de.intarsys.gars.core.demo.ui), "resource" is the name of the resource file without extension (e.g. messages) and "code" is the key in the properties file (e.g. MyButton.label).

Example

```
${nlsmsg.de.intarsys.gears.core.demo.ui.messages#MyButton.label}
```

This will look up the resource "messages_<lang>.properties" (respective "message.properties" if the "lang" code is not available) and resolve the property "MyButton.label".

If the resource or the property cannot be found, the message code itself is returned.

This lookup is not integrated with thymelaf.

8.2 Additional resource path

Normally resources are looked up in the JAR files of the runtime. To ease management of NLS files, for resolving an additional directory is added to the lookup.

The directory is denoted by "<appfamily>.i18n.dir" which defaults to "\${<appfamily>.config.shared}/i18n.

In this directory the resources are looked up according to Java package rules.

You can set the directory by adding the property to your configuration.

```
is-csc.i18n.dir=/opt/test/messages
```

You can use message definitions in this directory to **overwrite** existing messages. If a bundle is found both in the additional resource path and built-in, the additional bundle takes precedence. As both resource files are merged, you can create a partial redefinition, too.

9. Appendices

9.1 Cheat sheet

9.1.1 Windows locations

Variable	Description
<appfamily>.config.name	<appname>
<appfamily>.config.user	%USERPROFILE%/<appfamily>/config
<appfamily>.config.shared	%ProgramData%/<appfamily>/config
<appfamily>.data.user	%USERPROFILE%/<appfamily>/data
<appfamily>.data.shared	%ProgramData%/<appfamily>/data
<appfamily>.temp.dir	%AppData%/local/temp
<appfamily>.log.dir	%ProgramData%/<appfamily>/log
<appfamily>.log.name	<appname>

9.1.2 Linux locations

Variable	Description
<appfamily>.config.name	<appname>
<appfamily>.config.user	<user home>/<appfamily>/config
<appfamily>.config.shared	/etc/<appfamily>
<appfamily>.data.user	<user home>/<appfamily>/data
<appfamily>.data.shared	/var/lib/<appfamily>

<code><appfamily>.temp.dir</code>	<code>/tmp</code>
<code><appfamily>.log.dir</code>	<code>/var/log/<appfamily></code>
<code><appfamily>.log.name</code>	<code><appname></code>

9.1.3 Property definitions

Property files are read from these directories

- `${<appfamily>.config.shared}`
- `${<appfamily>.config.user}`

Valid property files are

- `${<appfamily>.config.name}.properties`
- `${<appfamily>.config.name}-<profilename>.properties`

9.1.4 Bean definitions

Bean definition files are not supported for this implementation.

9.1.5 Logging

The internal logback definition can be overridden by placing a logback.xml file at

- `${<appfamily>.config.shared}`
- `${<appfamily>.config.user}`

To only change the directory, use this property

```
<appfamily>.log.dir=/srv/logs
```

To only change the filename, use this property

```
<appfamily>.log.name=foobar
```

To only change the level, use this property.

```
<appfamily>.log.level=DEBUG
```

9.1.6 Licenses

Licenses are looked up at

- `${<appfamily>.config.shared}/licenses`
- `${<appfamily>.config.user}/licenses`

9.2 Test environment

For integration test purposes, Adobe Sign uses a well-known user

???

that is authenticated via "auth/login" (even if not supported officially).

10. External References

- [1] Adobe, „Electronic Sealing via customer-owned digital certificate,“ [Online]. Available: <https://helpx.adobe.com/sign/using/adobesign-pdf-digital-signer-api.html>.
- [2] Adobe, *Adobe TSP Partner Newsletter*, Jan, 2023.
- [3] csc, Architectures, Protocols and API Specifications for Remote Signature applications, v0.1.7.9.
- [4] Bundesdruckerei, Interface specification for sign-me API, v2.5.
- [5] csc, Architectures and Protocols for Remote Signature applications, v1.0.3.0.
- [6] Adobe, Adobe Acrobat Sign CSC API v1 Implementation Profile, v7.
- [7] Adobe, Adobe Sign implementation of the CSC API, v6.