Kofax SignDoc Web

Administrator's Guide

Version: 2.2.0

Date: 2019-05-22

KOFAX

© 2019 Kofax. All rights reserved.

Kofax is a trademark of Kofax, Inc., registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Kofax.

Table of Contents

Preface 5
Related documentation 5
Offline documentation 6
Training 6
Getting help with Kofax products 6
SignDoc Web features 7
Chapter 1: Functional structure of SignDoc Web 10
Architecture 10
Important files and directories 11
Communication with SignDoc Web server 15
Chapter 2: SignDoc Web installation as a Windows service 16
Definitions 16
General prerequisites 16
Quickstart 16
Content of the SignDoc Web ZIP archive 17
Production setup 17
Advanced configuration 19
Advanced information 20
Logging configuration 20
Installation on other JEE compliant application servers 21
Installation on Tomcat without using the provided service installer 21
Upgrade SignDoc Web 22
Upgrade from SignDoc Web 2.1 to 2.2.0 22
Upgrade from SignDoc Web 5.2.1 (or earlier) to SignDoc Web 2.1.0 22
Chapter 3: Deployment on Linux 23
Chapter 4: Administration and configuration 24
Alternative signature capturing via JavaScript - HTML5 data capture 24
Audit trail 27
Caching custom images 30
Check app environment 31
Click to sign 36
Cluster environment 38
Configuration of toolbar and GUI 39
Configure server 40

Custom CSS configuration....42
Digital signature certificate....44
Dynamic tablet screens....46
Encrypt sensitive data in sdweb_config.groovy configuration file....50
External help....51
Font mapping configuration....51
Gestures on mobile devices....52
Handling self-signed SSL certificates using iPads....54
iPhone configuration program....54
Download and installation....55
Create configuration profile....55
Install configuration profile on the iPad....62
Logging....64
Managing key pairs for encryption of biometric data....66
Multi-instance configuration....68
Page pre-fetching in WebView....69
Reduce network data....71
Sign2Phone administration....76
Signer-specific certificates....77
Kofax print plugin integration....78
TSA functionality....80
Additional notes....82
Chapter 5: Standard plugins....83
File DMS plugin....83
SFTP DMS plugin....88
Servlet DMS plugin....90
BasicAuthenticator plugin....92
Preload plugins....94
Chapter 6: Configuration file sdweb_config.groovy....96
Chapter 7: Frequently asked questions....119

Preface

SignDoc Web is a strategic enterprise e-signature platform. In general, SignDoc Web is a simple and straightforward to integrate PDF signing solution which can be used easily to replace existing paper-based processes. The product SignDoc Web offers web-based signing using handwritten signature, click-to-sign, or image/photo capture.

A prepared and prefilled PDF document is loaded into the browser and can be enriched with additional data. The handwritten signature which is added to the document will be captured on one of the available signature capture devices (e.g. sign pad, tablet PC, mobile device). Signature capture devices can be either connected to the PC directly or in the case of a smartphone for example accessing SignDoc Web directly. SignDoc Web supports also using the browser build-in HTML5 capture feature.

- During the signing ceremony, the biometric characteristics of the signer's signature are collected. With each captured signature time and date when the signature was captured will be stored together with it.
- As an alternative or in addition to the handwritten signature it is also possible to capture photos through a web or integrated camera and add them to the document.
- A third option is a click-to-sign signature which is simply the entering of the signers name in a text field showing a legal consent.

Upon saving or downloading the document the integrity value (hash) of the document will be calculated and stored in the signature field together with the biometric characteristics of the captured signature. The biometric data of the signature (e.g. coordinates, pressure, acceleration) is encrypted via the customer's public key. The biometric signature information can easily be decrypted with the customer's private key and displayed in a user friendly way via SignAlyze. All the changes and operations on a document are captured via an audit trail feature which is saved together with the document as metadata.

SignDoc Web also offers additional capabilities such as sending documents to an archive system and the possibility to be extended via a flexible plugin interface. The SignDoc Web application helps to minimize the footprint on the client side, since most of the software components are installed centrally on the server. The clients only need to have the signature capture device attached and the SignDoc Web Device Support installed. In addition the clients need a browser extension installed which provides the access to the capture device from SignDoc Web.

Related documentation

The full documentation set for SignDoc Web is available at the following location:

https://docshield.kofax.com/Portal/Products/en_US/SD/2.2.0-aq3yp7tk0j/SD.htm

In addition to this guide, the documentation set includes the following items:

- *Kofax SignDoc Web Help*
- *Kofax SignDoc Web Developer's Guide*

Offline documentation

Customers who require offline documentation can download the KofaxSignDocDocumentation_2.2.0_EN.zip from the Kofax Fulfillment Site. The .zip file includes both help and print folders.

1. From the Kofax Fulfillment site, download the documentation .zip file.
2. Extract the contents of the compressed documentation file.
3. Navigate for online help to folder help. Navigate for all other documentation to folder print.

Training

Kofax offers both classroom and computer-based training that will help you make the most of your Kofax SignDoc solution. Visit the Kofax website at www.kofax.com for complete details about the available training options and schedules.

Getting help with Kofax products

The Kofax Knowledge Base repository contains articles that are updated on a regular basis to keep you informed about Kofax products. We encourage you to use the Knowledge Base to obtain answers to your product questions.

To access the Kofax Knowledge Base, go to the Kofax website and select **Support** on the home page.

Note The Kofax Knowledge Base is optimized for use with Google Chrome, Mozilla Firefox or Microsoft Edge.

The Kofax Knowledge Base provides:

- Powerful search capabilities to help you quickly locate the information you need.

  Type your search terms or phrase into the **Search** box, and then click the search icon.
- Product information, configuration details and documentation, including release news.

  Scroll through the Kofax Knowledge Base home page to locate a product family. Then click a product family name to view a list of related articles. Please note that some product families require a valid Kofax Portal login to view related articles.
- Access to the Kofax Customer Portal (for eligible customers).

  Click the **Customer Support** link at the top of the page, and then click **Log in to the Customer Portal**.
- Access to the Kofax Partner Portal (for eligible partners).

  Click the **Partner Support** link at the top of the page, and then click **Log in to the Partner Portal**.
- Access to Kofax support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.

  Scroll to the **General Support** section, click **Support Details**, and then select the appropriate tab.

SignDoc Web features

Browser-independent

Simple web-based user interface. Support for most common browsers on PCs, such as Internet Explorer, Firefox, Chrome, Edge.

Content protection

Protect the integrity of documents by sealing them with a digital signature. Supports any x509v3 certificate provided through the key store (e.g. Windows certificate store) or through plugin interface to digitally sign documents by using an external method (e.g. for usage of a Hardware Security Module - HSM).

Customer-specific data

Set customer metadata for workflow process.

Customer-specific document processing

Use the plugin interface to write your own plugins for document processing.

Customer-specific labels

Customize labels of signature fields depending on the document workflow.

Customization

Enterprises can utilize SignDoc Web and launch their own e-signing solution, as well as integrate the functionality of SignDoc Web into their own apps (SDK available). Using your enterprise .css style is also supported.

Additionally, for user attendance to most important document content signature device (e.g. sign pad devices) background can be dynamically updated with document specific information.

Device-independent

Supports many different signature pads, interactive pen displays, tablet/slate PCs, iPad, Android and Windows tablets and iOS, Android and Windows smartphones.

Document binding

When a signature is captured it is safely embedded using an asynchronous public key encryption into and uniquely bound to the target document. Copy/paste attacks can thus be easily detected. SignDoc Web combines handwritten signatures with Public Key Infrastructure.

Enter signature fields

Enter signature fields anywhere on a document. It is useful for signing non-fillable documents.

Fill out and sign PDF forms

Complete and sign opened PDF forms or pre-populate the fields automatically.

Flatten PDF

Content in form fields is locked so it can be assured that information may no longer be changed. Flatten a PDF removes any layers (e.g. annotations, digital signatures) and consolidates them into one layer, which is supported by all PDF viewers.

Guidance in the signing process

Define and position data or signature fields and specify their completion/signing order. Additionally, highlight mandatory signature fields, define the order in which forms have to be signed, enforce the signing method, and much more. It is possible to disable certain functionality for a particular document, such as deleting a predefined signature field.

Handwritten signature capturing

A handwritten signature captured with SignDoc Web is much more than just an electronic image of a digitized signature embedded in a PDF or TIFF document.

SignDoc Web records - forensically identifiably - the handwritten signature of a person using all available parameters, such as writing movement, time, velocity, and acceleration.

Identity/signature verification

SignDoc Web captures the signature of a person using all available parameters of writing movement. If there is a doubt about the signature, an expert tool is available to forensically analyze the biometric characteristics of the captured signature. This capability can be taken one step further, with real-time verification of an acquired signature against a signature reference stored in a database to ensure that only authorized people can actually sign a document. Signature validation can be triggered for specific signature fields.

Integration

Integration with ERP, CRM, DMS, workflow management, etc via web services (SOAP, REST), Citrix and Terminal Server support.

For example: No need to transfer a PDF document. Users can receive a link to access the PDF on the server and only image previews are transferred to the app. Thus, the signed original document is securely server-based and not automatically copied (i. e. duplicated) to the mobile device. All manipulations of the PDF are always performed in the safe data center environment.

Offline

Take documents offline.

On-premises or cloud-based

SignDoc Web is available as an on-premises installation or can be installed as a cloud-based solution.

PKI-based certificates

Verify an electronic document before it will be signed to know if it is valid.

Print

Print a document before or after saving or signing.

Signature capturing

Sign electronic documents (PDF, TIFF) using handwritten signatures, photos or click-to-sign signatures.

Use PDF templates

Pre-populate, complete and sign PDF forms created from a template.

Watermark

Add watermarks like 'Confidential' or 'Draft' to your documents.

Chapter 1

Functional structure of SignDoc Web

This chapter provides insights into the functional structure including the various SignDoc Web components and configurations.

Architecture

Desktop clients

A variety of browsers are supported as desktop clients on Windows via the Kofax SignDoc Device Support in combination with a browser extension.

As an alternative to using a USB cable bound signpad for capturing the signature, on Windows it is also possible to use an Android smartphone or iPhone with the Sign2Phone app.

The SignDoc Web Remote Interface can be used to have full flexibility in embedding SignDoc Web into a customer´s web application i.e. by using an iFrame.

Mobile clients

SignDoc Web can be accessed via an iPad or Android tablet by using the SignDoc Mobile app from the respective app store. The SignDoc Mobile app allows the displaying and editing of documents to a certain extent as well as the capturing of the signature.

If the capturing of the signature should be performed via a pure browser-based environment on the mobile clients, it is also possible to enable HTML5-based signature capturing.

As with the desktop clients the Remote Interface is also available with the mobile clients to embed the SignDoc Web document frame into a customer´s web application.

Note No matter if the Remote Interface is used or not, when SignDoc Web is embedded into a customer's web application by iFrame, both web applications must be in compliance with the same-origin policy. How this can be achieved and how the same-origin policy can be relaxed is described in *SignDoc Web Developer's Guide,* chapter "Same-origin policy".

SignDoc Web server

Layer 1 - Operating systems:

SignDoc Web can be installed on Windows and Linux operating systems with a 32 or 64bit architecture

Layer 2 - SignDoc Web application:

The application consists of a WAR (Web Application Archive) which is deployed into a Web Application Server. The WAR file contains Native Libraries which are used for licensing and PDF handling. All the configuration files are stored outside of the WAR file in a folder called SDWEB_HOME. SignDoc Web also offers a plugin interface for possible means of extension.

Layer 3 - SOAP & REST interfaces:

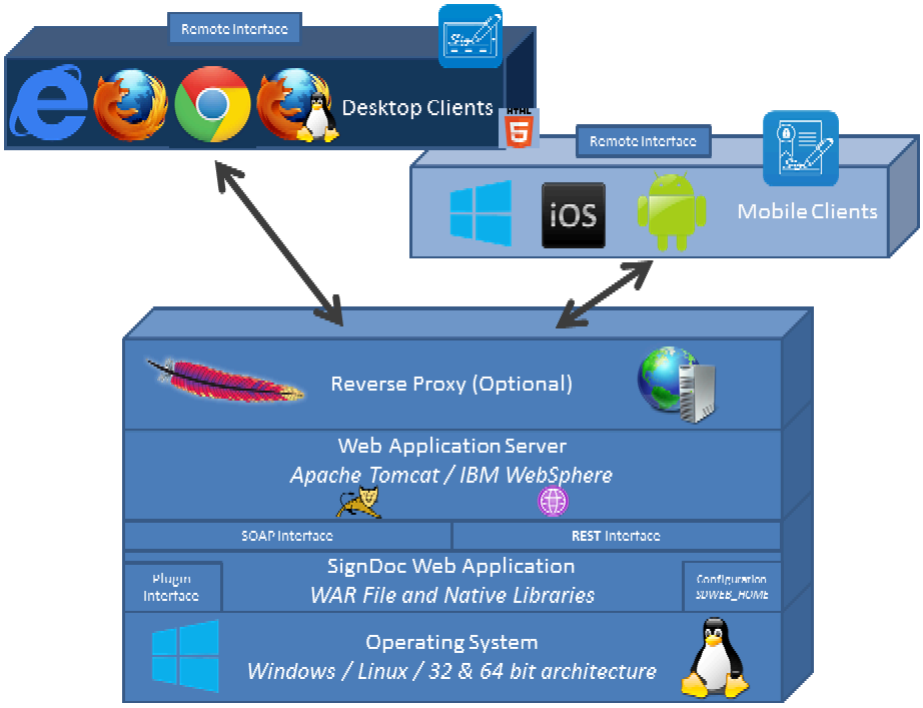
Documents can be uploaded via a web service and then accessed by using a Reference URL. It is also possible to upload and perform operations on documents via a REST interface.

Layer 4 - Web application server:

SignDoc Web runs on the standard Web Application Servers used within the industry: Apache Tomcat, IBM WebSphere. For supported versions and prerequisites, see *Kofax SignDoc Technical Specifications* document

Layer 5 - Reverse proxy (optional):

The usage of a Reverse Proxy (e.g. Apache HTTP or Microsoft IIS) is possible with SignDoc Web.



Important files and directories

SignDoc Web consists of two parts:

1. A WAR (Web application ARchive) file which needs to be deployed into a Web Application Server
2. The so called SDWEB_HOME directory which holds configuration and work data

Directory SDWEB_HOME

In this section we want to have a look at the files and directories which are provided as part of the **SDWEB_HOME** directory.

Directory	Description
c2s	Click-to-Sign templates
conf	Configuration files
dms	File-based DMS directory (created upon first use)
dms/ de.softpro.sdweb.plugins.impl.FileDms	SOFTPRO file DMS directory
doctemplates	Document templates
fonts	Font configuration
i18n	Global translation files (Internationalization)
interfaces	Plugin interfaces, examples and documentation
logs	Log files (audit, error, performance)
plugins	Non-Kofax plugins
resources	Resource directory for public key to encrypt biometric data of signatures and other configurable elements like icons.
tablet_screens	Tablet Screens (Backgrounds)
tools	TabletScreenImageCreator and PasswordEncryptionHelper

Directory conf

The **conf** directory contains the configuration files of SignDoc Web. Find below some information on the various files that are part of this directory.

Note If there is a need to run multiple SignDoc Web servers on one machine and configure them separately, have a look at the section Instance-based Configuration in the *SignDoc Web Administrator's Guide*.

File	Description
cert_store.p12	A custom PKCS#12 file containing a certificate that is used for the digital signing of the PDF document.
cert_store_readme.txt	Readme file explaining the usage of certificates.
configuration.xml	This is a bootstrap configuration file of the rich client which shouldn't be changed by a customer.
language.properties	This is the default language properties file used by the rich client if no language file is available for the locale.
language_xx.properties	These files contain all labels and messages which are visible in the rich client. The actual language file is chosen by the Java locale mechanism.
mobile_configuration.xml	In this rich client configuration file the functionality and features of the mobile GUI can be configured.

File	Description
sdcustom.properties	In this rich client configuration file the client logging and the supported locales can be configured.
sdweb_config.groovy	The sdweb_config.groovy file is the main configuration file for the server. It is written in a compact and structured syntax. It can be used in an instance-based configuration scenario to have a different server configuration for each instance.
server_configuration.xml	This is a bootstrap configuration file of the rich client which shouldn't be changed by a customer.
signdoc_configuration.xml	In this rich client configuration file the functionality and features of the desktop GUI can be configured.
XjLog.xml	A standard XjLog configuration file that can be changed for customized logging output. It can be used in an instance-based configuration scenario to have a different server configuration for each instance.

Directory doctemplates

The **doctemplates** directory contains various document templates which can be loaded in SignDoc Web. Find below some information on the various files that are part of this directory.

Note If new files are placed in the doctemplates directory a SignDoc Web server restart will be required for them to become available.

File	Description
lorem_ipsum_4_pages.pdf lorem_ipsum_contract.pdf softpro_banking_trapeza_de.pdf softpro_banking_trapeza_en.pdf softpro_contact_de.pdf softpro_contact_en.pdf softpro_photo_letter_de.pdf softpro_photo_letter_en.pdf softpro_sales_contract_de.pdf softpro_sales_contract_en.pdf TrapezaOpenJointAccounts.pdf	Sample PDF documents with generic capture and/or image capture fields.
lorem_ipsum_4_pages.pdf.base64 lorem_ipsum_contract.pdf.base64 TrapezaOpenJointAccounts.pdf.base64	Base64 representation of some of the sample PDF documents. This format can be used for the preload and prepare web service.
lorem_ipsum_contract.odt	Open Document file of the lorem_ipsum_contract.pdf document. This document can be loaded as a sample to see how text fields and checkboxes can be added to a document with Open Office.

Directory interfaces

The **interfaces** directory contains various SignDoc Web sample plugins and their documentation. Find below a brief overview of the sample plugins available.

Note More information on the sample plugins can be found in the *Kofax SignDoc Web Developer's Guide*.

Plugin	Description
NBC2SSignatureRendererSample	An example of an alternative Click-to-Sign image renderer.
PreparePlugin	A simple SignDoc Web Prepare plugin. The plugin automatically inserts a signature field in the bottom left corner of the first page. Additionally to this, it will populate each text field with the current date and check each checkbox form element.
ReadOnlyDmsPlugin	Simple DMS plugin demonstrating, how to make interactive fields readonly before storing the document in the DMS.
SetDocIdPlugin	This DMS plugin demonstrates, how a DMS plugin can set the document id.
SignatureArchivePlugin	A simple Signature Archive plugin, that stores all captured biometric signatures in the file system using the user id as index.
SignatureVerifcationPlugin	A plugin demonstrating the possibility to capture a signature reference and afterward's validating against it.
SignPKCS7Demo	A plugin demonstrating the possibility to sign documents using an external method.
VSVTestPlugin	Combined Prepare and Signature Verification plugin, that will display a dialog with the captured signature and the signature of Albert Einstein as reference signature.
ZDms	A simple DMS plugin saving documents to a directory.
ZPrepare	A prepare plugin which inserts signature fields based on location, textphrase and parts of the document id.

Directory logs

The **logs** directory contains the SignDoc Web log files and subdirectories. Find below a brief overview of log files and directories available.

File / Directory	Description
audit\generic	Subdirectory holding audit logs which contain general information like document opened, signed per user.
audit\transactions	Subdirectory holding audit logs which contain transaction information per user.
audit\users	Subdirectory holding audit logs per user.
error	Subdirectory holding error logs.
sdweb.log	The default SignDoc Web log file.

File / Directory	Description
sdweb.log.x	Log rollover of sdweb.log file.
Statistic.log	The statistics log file.

Directory tablet_screens

The **tablet_screens** directory contains the tablet background screens that can be used by SignDoc Web. Find below a brief overview the files in this directory.

File	Description
default.xml dynamic_content_example.xml next_screen.xml ok_screen.xml piggybank_example.xml WacomSTUSeries.xml	Sample Tablet Screens available with SignDoc Web.
dynamic_content_example.html piggybank_example.html Piggybank_200.png	Test pages to test Dynamic Content displayed on the signpad.
TabletScreenLayout.xsd	XML Schema of Tablet Screens

Communication with SignDoc Web server

There are various ways of communicating with the SignDoc Web server:

- Desktop client: The communication with the SignDoc Web server takes place via the browser directly
- Mobile client: The communication with the SignDoc Web server takes place via the iOS or Android app or the browser directly (HTML5 capturing)
- Web service: The communication with the SignDoc Web server takes place via the SOAP protocol
- REST: The communication with the SignDoc Web server takes place via HTTP calls

Chapter 2

SignDoc Web installation as a Windows service

This chapter provides information on how to install and configure SignDoc Web as a Windows service.

Definitions

- `INSTALLDIR` is the directory of the unpacked SignDoc*-tomcat.zip file. See Quickstart procedure.
- `SIGNDOC_HOME, SDWEB_HOME` are different names for the home directory of SignDoc Web. See Content of the SignDoc Web ZIP archive, section "Directories".

General prerequisites

Before starting the installation it is required to install and check the following prerequisites.

To be able to use SignDoc Web 2.2.0, you have to install these Microsoft Visual C++ Redistributable Packages:

- Visual Studio 2008 SP1
- Visual Studio 2017

  Depending on the Windows version, it might be necessary to install some updates via Windows Update before this setup can be successfully installed.
- Windows PowerShell (powershell.exe) must be in the system path. This should be the normal case for the Windows Server OS.
- If this is an upgrade of SignDoc Web, check Upgrade SignDoc Web.

Note It is recommended to install SignDoc Web behind a reverse proxy. If the reverse proxy is also used to load-balance requests, it can be done stateless (e.g. round-robin).

Quickstart

Getting a simple local accessible SignDoc Web installation running can be achieved in less than 5 minutes. It is not wasted time doing this, since it is a base for a production ready setup.

Quickstart goals

- Install SignDoc Web as a Windows service.

Quickstart prerequisites

- 4 GB RAM

Quickstart procedure

- Unpack the signdoc-web*-tomcat.zip file in a new directory `INSTALLDIR`.
  Example
  `c:\Program Files\signdoc-web-2.2.0`
- Double-click `INSTALLDIR\service_up.cmd`
- Wait approximately 1 minute on first start.
- Open SignDoc Web: http://localhost:6610/sdweb.

Content of the SignDoc Web ZIP archive

The relevant and configurable content of SignDoc Web consists basically of 3 files:

- a configuration file
- a script to install and configure the SignDoc Windows service
- a script to deregister the SignDoc Windows service

Tools

- `INSTALLDIR\service_up.cmd` installs, applies configuration and restarts the SignDoc Windows service.
- `INSTALLDIR\service_remove.cmd` stops and deregisters the SignDoc Windows service. No files are deleted.
- `INSTALLDIR\service_configuration.properties` is the configuration file of the SignDoc Windows service. This file can be edited with a regular text editor. The syntax and usage is described in the file.

Directories

- `INSTALLDIR\signdoc_home` is the default SIGNDOC_HOME / SDWEB_HOME directory.

Production setup

The following chapters describe basic tasks that should or must be completed for a production setup.

Goals for production

- Configure network settings
- Configure reverse proxy setup (optional)
- Advanced configuration (optional)

Prerequisites for production setup

- Application server
  - min. 2 GB RAM. See Advance configuration, section "Tune Java memory settings".
  - min. 2 GB free disk space

- SignDoc Web
  Install SignDoc Web as described in Quickstart.

Procedure for production

The following topics do not depend on each other and can be executed independently. The settings are applied by double-clicking INSTALLDIR\service_up.cmd.

Configure network settings

SERVICE_HTTP_PORT

Open `INSTALLDIR\service_configuration.properties` in a text editor.

Navigate to `SERVICE_HTTP_PORT`.

Set the preferred port number.

Example

```
SERVICE_HTTP_PORT=6611
```

SERVICE_EXTERNAL_HOST_URL

A production service must be accessible via official domain name, so it can be accessed from other computers. See section "Configure reverse proxy setup".

SERVICE_INTERNAL_HOST_URL

This is the context URL that is used to transmit data that does not have to be routed over public networks.

> **Important** Usually this setting should not be changed.
>
> Exception: If the Tomcat server is configured for "HTTPS only" connections, the URLs scheme must be changed from http to https. The default setting that is based on localhost is usually correct.

Example

```
# ${SERVICE_HTTP_PORT} will be replaced by service_up.cmd with the corresponding value.
SERVICE_INTERNAL_HOST_URL=http://localhost:${SERVICE_HTTP_PORT}
```

Configure reverse proxy setup

In a reverse proxy scenario it is important to configure the application URLs correctly. There are 2 URLs to configure.

Open `INSTALLDIR\service_configuration.properties` in a text editor.

Navigate to the settings below.

Change the values if required.

SERVICE_EXTERNAL_CONTEXT_URL

This is the context URL that is used to access the application. This URL must be reachable from anywhere and is part of the signing links that are sent via email.

Example

```
SERVICE_EXTERNAL_CONTEXT_URL=https://signdoc.mydomain.com
```

Advanced configuration

General

To configure more features of SignDoc Web, there are 2 options:

1. Edit the file:
   `INSTALLDIR\_conf_templates\sdweb_config.groovy`
2. Apply the configuration by executing `INSTALLDIR\service_up.cmd`.

   Note The configuration defined in `INSTALLDIR\service_configuration.properties` takes precedence over settings with the same name in cirrus.properties.

Tune Java memory settings

For full details, see https://tomcat.apache.org/tomcat-8.5-doc/windows-service-howto.html.

The default Java memory settings for SignDoc Web are

```
JvmMs=1024
```

(minimum heap = 1GB)

```
JvmMx=2048
```

(maximum heap = 2GB)

If these settings should not be sufficient, open `INSTALLDIR\service\bin\signdoc_service_install_as_admin.cmd` in a text editor.

Navigate to

```
--JvmMs=1024 ^
```

```
--JvmMx=2048 ^
```

Change the settings.

Restart the server with `INSTALLDIR\service_up.cmd`.

Example for 4GB maximum heap

```
--JvmMx=4096 ^
```

Use SignDoc in a clustered environment

SignDoc can be used in a clustered environment. A typical use case is load balancing.

Requirement: A SignDoc load balancer must support application defined session affinity. I.e. for one particular singing session, subsequent requests must be routed back to the same instance. This is archived by respecting the JSESSIONID cookie that Tomcat sets in HTTP responses.

To achieve high availability for the signing sessions the tomcat server must be configured to cache the sessions in central location. Typically, this is achieved with using separate session database or simply by a memcached instance. This configuration must be applied separately to the tomcat configuration.

If multiple instances should be installed on the same operating system, it must be ensured to use a different HTTP/TCP port for each instance. See Production setup, section "Configure network settings". The default HTTP/TCP port for SignDoc Web is 6610.

Note If the server.xml file must be changed, it should be done in the file `INSTALLDIR\_conf_templates\server.xml`. Execute service_up.cmd to apply the change. See Content of the SignDoc Web ZIP archive, section "Tools".

Advanced information

View service details

To view a detailed service information double-click

`INSTALLDIR\bin\SignDocWeb.exe`

It is not recommended to change settings with this tool, since they are being overwritten whenever service_up.cmd is executed. See Content of the SignDoc Web ZIP archive, section "Tools" and Advanced configuration.

Logging configuration

The logging configuration is defined in the file

`INSTALLDIR\signdoc_home\conf\XjLog_sdweb.xml`

and can be edited.

The default location of the XjLog_sdweb.xml file is

`INSTALLDIR\signdoc_home\conf\`.

Internal Tomcat logging

The internal Tomcat logging is turned off by default. It can be re-enabled if desired by uncommenting the line starting with

`#handlers`

of file

`INSTALLDIR\_conf_templates\logging.properties`

Execute service_up.cmd to apply the change. See Content of the SignDoc Web ZIP archive, section "Tools").

Installation on other JEE compliant application servers

While this is generally technically feasible, this is not covered or supported by this guide.

Installation on Tomcat without using the provided service installer

SignDoc can be installed on an already existing Tomcat 8 or Tomcat 9 server. The following things must be considered:

- You must use
  - the latest Tomcat 8.5.x or Tomcat 9.x server
  - the latest Java 8 64 Bit JRE to run the Tomcat 8.5.x server. Java 11 JRE is supported for Tomcat 9 server.
- Unpack the sdweb.war file manually in `%CATALINA_HOME%\webapps` so that there is a sdweb context directory
- Copy the following jar files from the `service\lib` directory to the `lib` directory of the Tomcat directory (usually CATALINA_HOME):
  - splm2jni-*.jar
  - SPSignDoc-*.jar
  - SPSignware-*.jar
- Make sure that the directory with the native libraries `%CATALINA_HOME%\webapps\sdweb\WEB-INF\lib\native\Win64` is in the PATH of the Tomcat process
- Make sure that there are no older SignDoc native libraries in the PATH of the Tomcat process
- Copy the `signdoc_home` directory to the desired location. This location must be readable and writable for the Tomcat process.
- Copy `_conf_templates\sdweb_config.groovy` to the `signdoc_home/conf` directory and configure it as desired.
- To allow big file uploads it might be required to adjust the maxPostSize attribute of the <Connector> element of Tomcat's server.xml
- Start the Tomcat service with the following system properties:
  - Required properties:

    ```
    -DSDWEB_HOME=<path_to_signdoc_home_directory>
    -DSERVICE_INTERNAL_HOST_URL=<loopback_url_to_root_context>
    -DSERVICE_EXTERNAL_HOST_URL=<generally_accessible_url_to_root_context>
    ```

    Example:

    ```
    -DSDWEB_HOME="c:/signdoc_home"
    -DSERVICE_INTERNAL_HOST_URL=http://localhost:8080
    -DSERVICE_EXTERNAL_HOST_URL=http://mysigndocserver.example.com
    ```

Upgrade SignDoc Web

Upgrade from SignDoc Web 2.1 to 2.2.0

- Stop SignDoc Web 2.1 using service_remove.cmd. See Content of the SignDoc Web ZIP archive, section "Tools".
- Install SignDoc Web 2.2.0 as described in Quickstart above.
- Apply the existing SignDoc Web 2.1 configuration to the new SignDoc Web 2.2.0 installation. This means, applying any existing configuration from service-configuration.properties (and possibly other configuration files) to the new installation. This usually concerns among other things: SERVICE_EXTERNAL_HOST_URL, ...
- Optional: Copy/apply old configuration settings (e.g. sdweb_config.groovy) to the new SDWEB_HOME
- Start the new SignDoc Web 2.2.0 version using service_up.cmd. See Content of the SignDoc Web ZIP archive, section "Tools".

Upgrade from SignDoc Web 5.2.1 (or earlier) to SignDoc Web 2.1.0

To upgrade an existing SignDoc Web 5.2.1 (or earlier) to 2.1.0 perform the following steps:

Stop and disable automatic restart for all existing SignDoc Web instances.

Remove and uninstall prior installations of SignDoc Standard or SignDoc Web. Also make sure to remove all SDWEB_HOME and/or CIRRUS_HOME directories.

Install SignDoc Web 2.1.0 as described in Quickstart.

Configure SignDoc 2.1.0 for production as described in Production setup. It is recommended to apply the configuration fresh.

Start the new SignDoc Web version using service_up.cmd. See Content of the SignDoc Web ZIP archive, section "Tools".

Open http://\<your_server\>:\<port\>/sdweb

Chapter 3

Deployment on Linux

If SignDoc is to be deployed under Linux, it is recommended to use Docker. SignDoc provides several sample Dockerfiles (in the docker directory). These Dockerfiles contain documentation and describe the basic installation procedure for a Linux environment. These files can also be used and amended/adapted for deployment, if required/desired.

Chapter 4

Administration and configuration

The following chapter gives general information and tips on how to administer and configure SignDoc Web.

Alternative signature capturing via JavaScript - HTML5 data capture

Important The SignDoc Web HTML5 capture feature enables the acquiring of signature time series data via a browser client directly from the device without additional installation of software components. It should be noted that capturing data via HTML 5 does not currently provide pressure information.

The SignDoc Web HTML5 capture is available for the following browser applications: Safari, Internet Explorer, Firefox, Chrome, Edge, and Opera. Note that signature quality (both visual and data content) will vary between different browsers and device types.

Limitations you should be aware of when using HTML 5 capturing are:

- The captured data are x and y screen coordinates, and time.
  Time between two samples is not equidistant.
- Currently, signatures are recorded with a fixed 96 dpi entry due to technical limitations.
  Some devices have different X and Y resolutions, which can stretch signatures on one axis.
- The browser used will affect the number of sample points.
  The rate of captured sample points within time duration will have a strong variation depending on the device running the browser application.

Note HTML5 captured signatures are suitable for signing of documents with simple legal requirements, and provide technical integration convenience. The HTML5 captured signatures however may not be used for automatic signature verification, and provide very limited forensic evidence. This is primarily due to a lack of sufficient and reproducible characteristic data of the individual’s signature.

The mobile view is displayed by default in the SignDoc Mobile App which should be installed on the mobile device if SignDoc Web is used for document processing. Device (and OS) specific signature capturing is enabled within SignDoc Mobile.

Signatures can be captured in the desktop browser with the SignDoc Web client with the help of Kofax SignDoc Device Support in combination with a browser extension which allows 'native' access to the connected capture device. The desktop view is used in the browser for displaying a loaded document.

With the technical capabilities of HTML5 supporting browsers it is possible to capture signatures only with JavaScript.

With this method it is possible to capture signatures on mobile devices also without the necessity to use SignDoc Mobile App.

In the desktop browser you can capture signatures without any additional software installed on the machine.

It is configurable in

```
%SDWEB_HOME%/conf/sdweb_config.groovy
```

whether JavaScript capturing is used or not.

- **sdweb.capture.html5_mobile** Supported values: force, deny

  deny - JavaScript capturing cannot be used on mobile devices. SignDoc Mobile App must be installed if a browser loads a document in SignDoc Web.

  force - JavaScript capturing must be used, SignDoc Mobile App is not called from a mobile browser if a document is loaded with SignDoc Web.

  Default: deny
- **sdweb.capture.html5_desktop** Supported values: force, deny

  deny - JavaScript capturing cannot be used in a desktop browser. Kofax SignDoc Device Support in combination with a browser extension must be installed on the machine for capturing signatures in a document in SignDoc Web client.

  force - JavaScript capturing must be used in the desktop browser. The document is loaded in mobile view ([1]) and capturing via browser extension is disabled.

  Default: deny

([1]) For now JavaScript capturing is only enabled in the mobile view. That means that always the mobile document view is used for enabled JavaScript capturing independent from the client requester type which could be mobile or desktop.

Note These settings are overwritten when the SignDoc Web client is integrated by a webpage using the RemoteInterface (see also *SignDoc Web Developer's Guide*, chapter "RemoteInterface".

In browsers that don't support HTML5 functionality the SignDoc Web client tries to execute a flash plugin for capturing per default.

If flash plugins are not allowed in certain customer environments it is possible to disable it via configuration entry 'Flash.Allowed' (see configuration below).

If neither JavaScript capturing nor Flash capturing is possible in a browser a corresponding message is displayed to the user.

Configuration entries in mobile_configuration.xml that affects HTML5 capturing and desktop browser support:

Settings for the JavaScript capture dialog

```
<component id="SD_SignatureCaptureDialogHtml5">
<element id="Title">
```

```
<parameter key="Label" propertyFile="language"
 propertyKey="SD_SignatureCaptureDialog.Title"/>
</element>
<element id="Size">
<parameter key="Frame.Width" value="740"/>
<parameter key="Frame.Height" value="375"/>
<parameter key="Width" value="745"/>
<parameter key="Height" value="380"/>
</element>
<element id="Misc">
<!--
'false' - capturing is displayed in a simple popup without an explicit dialog close
 button.
'true' - capturing is displayed in a real dialog with an explicit dialog close button
 at the bottom.
-->
<parameter key="Closable" value="false"/>
<!--
Specify if the content of this dialog should be scrollable if it exceeds the dialogs
 size.
Setting this to false would make sense if the height of the dialog is reduced in order
 to hide the capture buttons on the screens.
Doing this the capture process can only be controlled by the tablet device.
-->
<parameter key="Scrollable" value="false"/>
<!--
Specify the pressure level which defines the thickness of the rendered signature in the
 document.
The recommended value is 1023 (valid range is from 1 - 1023).
-->
<parameter key="Pressure" value="1023"/>
<!--
Specify the text which should be displayed below the signature line.
-->
<parameter key="Text" propertyFile="language"
 propertyKey="SD_SignatureCaptureDialogHtml5.Text"/>
<!--
Specify if Flash plugin is allowed when browser doesn't support HTML5.
-->
<parameter key="Flash.Allowed" value="true"/>
</element>
</component>
```

Settings that allow the usage of the mobile-gui in environments containing both, desktop- and mobile browsers

Zoom settings

```
<component id="Lists">

<!-- The zoomlist configuration for browsers with touch support using the mobile gui.
     Hint: the zoomlist configuration for desktop browsers without touch support can be
 found below -->
<element id="ZoomList">
...
</element>

<!-- The zoomlist configuration for desktop browsers without touch support using the
 mobile gui. -->
<element id="ZoomList_Desktop">
...
</element>
```

```
</component>
```

Toolbar settings

```
<!-- The toolbar configuration for browsers with touch support using the mobile gui.
     Hint: the toolbar configuration for desktop browsers without touch support can be
 found at the end of this file -->
<component id="Toolbar">
...
</component>

<!-- The toolbar configuration for desktop browsers without touch support using the
 mobile gui -->
<component id="Toolbar_Desktop">
...
</component>
```

Audit trail

Description

SignDoc Web can record information about user actions while a document is processed. This information can be recorded in any language if the appropriate translation files are present.

The SignDoc Web Audit Trail is activated by configuring and enabling an IAuditLog plugin in SignDoc Web. A default Audit Trail Plugin de.softpro.sdweb.plugins.impl.SimpleAuditLog is included in SignDoc Web as core plugin.

Note Note This feature is especially important in a C2S environment.

Configuration

For example in the sdweb_config.goovy configuration file the following settings can be made:

```
sdweb.audittrail.enabled=true
sdweb.audittrail.plugin.impl="de.softpro.sdweb.plugins.impl.SimpleAuditLog"
sdweb.audittrail.locale="en"
sdweb.audittrail.generic=true
```

Options

- **sdweb.audittrail.enabled** (boolean): Defines, if the Audit Trail function is generally enabled and SignDoc Web should log the user actions. Default: true
- **sdweb.audittrail.plugin.impl** (string): The IAuditLog plugin to use. Default: de.softpro.sdweb.plugins.impl.SimpleAuditLog
- **sdweb.audittrail.locale** (string): The language to use to write the Audit Trail. Default: en
- **sdweb.audittrail.generic** (boolean): If set to true, a generic Audit Trail Log will be always written. Even if the uid (c2s_uid - c2s user id) and/or the tid (c2s_tid - c2s transaction id) is not set. See also Click to sign, section "HTTP servlet parameters".

Use SimpleAuditLog

The SimpleAuditLog plugin provides an Audit Trail logging implementation. The plugin logs the audit information in separate log files on the file system. The output format is CVS. An example output is listed below.

Note The default behavior is, that the logs will be written in the directory SDWEB_HOME/logs/audit.

Logfiles are created following this rule:

```
<logfile_dir>/<date>/<logfiles>

logfile_dir:=log direcory for the logfile_types transactions|users|generic
date:=the date of the log
logfiles:=depends on the logfile_type
```

Examples

Log type: Transactions

```
sdweb_home/logs/audit/transactions/2012-01-29/1234.log
```

where 1234 is the transaction id

Log type: Users

```
sdweb_home/logs/audit/transactions/2012-01-29/chi.log
```

where chi is the user id

Log type: Generic Log

```
sdweb_home/logs/audit/generic/2012-01-29/generic_audit.log
```

This log is a full audit log.

- Transaction Log (based on the servlet parameter c2s_tid)
  - Contains all log information about a transaction
  - For each unique transaction per day a new file is created: <c2s_tid>.log
- User Log (based on the servlet parameter c2s_uid)
  - Contains all log information of a particular user
  - For each unique user per day a new file is created: <c2s_uid>.log
- Generic Log
  - A new log file is created per day that contains all audit log information of the day
  - Can be explicitly enabled/disabled by the plugin configuration option: enable_generic_log (see below)

Example

```
sdweb_config.goovy
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".transaction_log_dir="/var/
softpro/log/transaction_logs"
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".users_log_dir="/var/
softpro/log/user_logs"
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".genric_log_dir="/var/
softpro/log/genric_log"
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".separator="|"
```

```
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".dateformat="yyyy-MM-dd"
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".timestampformat="yyyy-MM-dd
 HH:mm:ss"
sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog".enable_generic_log=true
```

Configuration options for SimpleAuditLog

(omitting the prefix sdwebplugins."de.softpro.sdweb.plugins.impl.SimpleAuditLog" for the Name attribute)

- **transaction_log_dir** (string): The path, where the transaction logs should be written to. Each transaction will create a new file with the transaction id as file name. To write a transaction log, the servlet parameter c2s_tid has to be set. See also: Click to sign, section "HTTP servlet parameters". Default: SDWEB_HOME/logs/audit/transactions/<current_date>
- **users_log_dir** (string): The path, where the user logs should be written to. Each user will create a new file with the user id as filename. To write a user log, the servlet parameter c2s_uid has to be set. See also Click to sign, section "HTTP Servlet parameters". Default: SDWEB_HOME/logs/audit/users/<current_date>
- **genric_log_dir** (string): The path, where the generic logs should be written to. The generic log is written, if sdweb.audittrail.generic is set to true (see above). Default: SDWEB_HOME/logs/audit/generic/<current_date
- **separator** (string): The contents of the file are organized as CSV. This string defines the separator string. Default: "|"
- **dateformat** (string): The date format to be used when writing the timestamps. The format must be compliant to the Java SimpleDateFormat class: docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html. Default: yyyy-MM-dd HH:mm:ss
- **enable_generic_log** (boolean): If set to true a generic log based on granularity of the dateformat String will be created containing all audit log entries. Default: true

CSV format for SimpleAuditLog

```
timestamp | transaction id | user id | action code | document id | localized
action message
```

Example

```
...

2012-01-24 15:18:20|1327413414893|chi@softpro.de|1|2012-01-24_15-18-20-902|
Document was opened|

...
```

How to develop customer-specific audit log

To develop an own customized plugin, it is recommended to use this as your basic class definition:

```
public class SimpleAuditLog extends AbstractPlugin implements IAuditLog {

@Override
public synchronized Map<String, Object> logEvent(AuditLogKey auditLogKey,
 AuditEventType auditEventType, final Map<IAuditLog.PLUGIN_PARAMS, Object> params)
 throws PluginException {
  // implementation code
}

// other required methods
```

```
}
```

For further details consult the provided JavaDoc in:

`SDWEB_HOME/interfaces/plugins`

Caching custom images

Image caching

It is already possible to enable caching for <%PRODUCT%> images in sdweb_config.groovy with

```
sdweb.view.imagecache.enable=true
```

This setting affects NOT the custom specific images which can be addressed via "/custom/img".

These custom images can be specified in sdweb_config.groovy via

```
sdweb.cache.custom_images.list=[..., ...]
```

The list can contain one or more regular expressions patterns.

If the client request matches with one of these patterns the request will be cached on the client.

Example

```
sdweb.cache.custom_images.list=[".*mobile_toolbar_.*\\.png",
".*desktop_toolbar_.*\\.png"]
```

This pattern causes all requests that contain 'mobile_toolbar_' and end with '.png' to be cached as well as all requests containing "desktop_toolbar_" and ending with ".png"

The max-age cache for these custom images can be configured (in seconds) via

```
setting sdweb.cache.custom_images.max_age=...
```

The default is 259200 for 3 days in seconds (3*24*60*60).

GWT files caching

GWT specific xxx.cache.html files are cached.

The http response header "Cache-Control" is set with "max-age=..."

for these files with

```
sdweb.cache.gwt.max_age=...
```

in seconds, default is 259200 (3*24*60*60) for 3 days

(sdweb.cache.gwt.max_age=259200)

Additional requests caching

It is possible now to enable the caching of additional specific requests which matches with one of the (regular expressions) patterns in the list

```
sdweb.cache.pattern.list=["...", "..."]
```

Example

```
sdweb.cache.pattern.list=[".*dark\\.css", ".*toolbar.*\\.png"],
```

which matches with either any requests ending with dark.css or with requests which have 'toolbar' anywhere in the URI and ends with '.png'.

Note It is also possible to define only one pattern in the list (without separating comma).

The http response header "Cache-Control" with "max-age=..." can be set for these files (in seconds) with

```
sdweb.cache.pattern.max_age=...
```

Default is 259200 (3*24*60*60) for 3 days in seconds.

Check app environment

Preface

With SignDoc Mobile it is possible to use SignDoc Web functionality via iOS or Android app.

It can be useful to restrict the access to specific apps or devices.

This can be reached by implementing either a so-called whitelist or a blacklist or maybe a mixture of both.

The whitelist enables only specific apps or devices to access SignDoc Web.

A blacklist rejects specific app or device configurations.

An app must pass some environment data to SignDoc Web for verification.

The user will be informed if the app or the device is either not allowed or deprecated whereas in the latter case he is able to continue nevertheless.

App environment data

The app has to send some environment data for verification to the SignDoc Web application server. The following information is included in the request:

AppVersion, ScreenSize, OS, AppID, DeviceModel, CPUModel, CPUClock, DeviceVendor, CaptureType and ConfigLinkID.

Blacklist and whitelist

A whitelist can be defined, in order to allow only specific apps or devices.

If an app or a device is not included with the specified features it is not allowed to use the SignDoc Mobile App.

The blacklist can reject specific apps or devices or can declare them as questionable, where the user gets only a warning message but he can continue processing with the app nevertheless.

It is possible to define several blacklist and whitelist sections. Each of them contains a specific combination of characteristics according to the requesting app and device.

The entries in such a section are compared with the characteristics which are sent by the app. A match only takes place if all of the entries in a section match with the characteristics of the app.

If both whitelists and blacklists are specified it is a little bit more complicated.

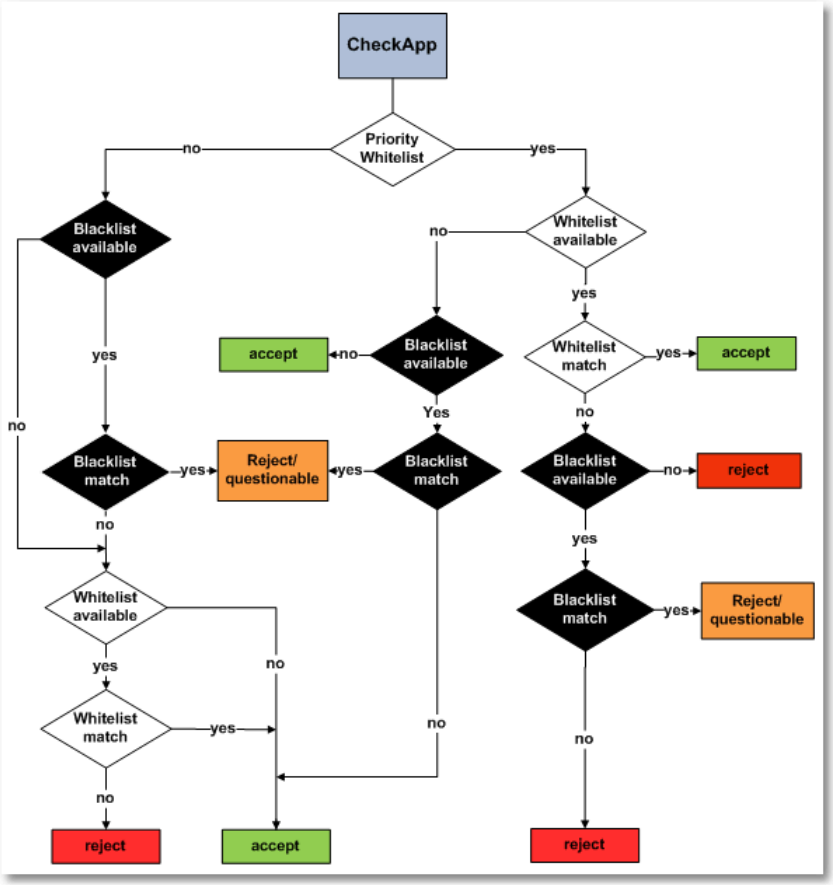
Per default the blacklist has a higher priority than the whitelist.

The priority can be changed in sdweb_config.groovy with the configuration setting:

```
sdweb.prioritizeWhiteList=true
```

Decisions according to blacklist and whitelist definitions

The following picture shows the decision flow chart if whitelists and blacklists are defined, dependent from the configured priority.



Blacklist and whitelist configurations

Blacklists and whitelists can be defined also in sdweb_config.groovy.

The configuration section of blacklists and whitelists for supported apps starts with id 'supportedApps'. The lists are included in curly brackets.

You can define either a whiteList section and/or a blackList section.

A whitelist section starts with "whiteList =" followed by one or more list entries included between square brackets and separated by a comma.

Each section has one or more entries of app or requesting device characteristics which must be verified. The following characteristics (corresponding to the app input information) can be listed:

AppVersion, ScreenSize, OS, AppID, DeviceModel, CPUModel, CPUClock, DeviceVendor, CaptureType, ConfigLinkID

The characteristics are written in format:

```
key : value [,]
```

whereas the key is one of the mentioned characteristics and the value is the string (in double quotes) to compare with.

Example

```
AppVersion : "1.2.0"
```

Several entries are separated by commas.

Note Only strings (double quoted) are allowed as values, even if they represent a number.

The blacklist has the same structure in principle but starting with "blackList =".

If a whitelist is specified and no match is found an appropriate return code is returned to the App together with a message which is displayed to the user.

The message is defined in sdweb.properties (part of WEB-INF\lib\sdweb_4.3.jar in deployed sdweb.war) by default with

```
sdweb.appenvcheck.nowhitelistmatch=The app or your device is not supported,
please contact your system administrator
```

Customer specific messages

The customer can change the message by using the customer specific message features of SignDoc Web.

With SignDoc Web 3.1.6 you can currently only enter customer specific (Server) messages in:

```
%SDWEB_HOME%/i18n/TextTranslationTable.xml
```

Sample for creating another message in TextTranslationTable.xml with another value for the key sdweb.appenvcheck.nowhitelistmatch:

```
<tns:translation_entry ID="sdweb.appenvcheck.nowhitelistmatch">
```

```
  <tns:default>
    <tns:value>The App is not allowed, please contact your system administrator
    </tns:value>
  </tns:default>
  <tns:translation lang="de">
    <tns:value>Die App ist nicht zugelassen, bitte kontaktieren Sie Ihren System
 Administrator
    </tns:value>
  </tns:translation>
</tns:translation_entry>
```

The general message after a reject caused by a match in a blacklist can be overridden by specific customer messages. Dependent from the App or device characteristics which should be rejected for blacklist section it makes sense to define different reject messages which are displayed to the user. In order to make a reject clearer for the user it is possible to integrate also reject values of the characteristics in the message, like:

```
Your App version '1.2.1' is too old, please update to current version.
```

This can be realized by defining placeholders for reject values in the message, which are replaced during runtime. The previous sample would be defined as

```
Your App version ${AppVersion} is too old, please update to current version.
```

All placeholders are embedded in curly brackets with a $ prefix. The placeholder name must exactly match the key definition of the characteristic in the black list section.

The default message for a questionable blacklist match is:

```
'The App or your device is deprecated, please update as soon as possible.'
```

A blacklist section can have two additional entries msgid and questionable.

The msgid defines the message identifier for the customer specific message. This message identifier is appended to the prefix "sdweb.appenvcheck.match.".

If you define "blacklist1" as msgid in your section you have to define your specific message under the translation entry sdweb.appenvcheck.match.blacklist1.

Example

```
<tns:translation_entry ID="sdweb.appenvcheck.match.blacklist1">
  <tns:default>
    <tns:value>Your ${DeviceVendor} device model ${DeviceModel} is not qualified
 with this ${OS} operating system, please buy another device which does not come from
 ${DeviceVendor}.</tns:value>
  </tns:default>
  <tns:translation lang="de">
   <tns:value>Ihr ${DeviceVendor} Gerät Modell ${DeviceModel} ist nicht
 zugriffsberechtigt mit dem Betriebssystem ${OS}, bitte besorgen Sie sich ein anderes
 Gerät, das nicht von ${DeviceVendor} hergestellt wird.
   </tns:value>
  </tns:translation>
</tns:translation_entry>
```

or sample for questionable return code

```
<tns:translation_entry ID="sdweb.appenvcheck.match.blacklist2">
```

```
  <tns:default>
    <tns:value>The App version ${AppVersion} is deprecated. It is recommended to update
 your App to the newest version. Press the Cancel button to stop processing or press
 the Ignore button to continue nevertheless.
    </tns:value>
  </tns:default>
  <tns:translation lang="de">
    <tns:value>Die App Version ${AppVersion} ist veraltet. Es wird empfohlen auf die
 neueste App zu aktualisieren. Drücken Sie den Abbrechen Button, um den Vorgang zu
 beenden oder drücken Sie den Ignorieren Button, um trotzdem fortzufahren.
    </tns:value>
  </tns:translation>
</tns:translation_entry>
```

Definition of blacklists and whitelists

The parameter questionable equals "true" indicates that a match is only a questionable reject not a final reject if the characteristics of the requesting App or device matches with all entries of the appropriate blacklist section.

In this case a specific "Questionable" return code is passed back to the App which enables the user to continue processing (with a warning or note) although the requesting App or device matches with the blacklist section.

Example of blacklist and whitelist definition in sdweb_config.groovy:

```
  // configuration section of supported apps
supportedApps {
  // The list of supported Apps can be configured either as whitelist
  // which contains all allowed App environments
  // or Apps can be excluded (also questionable) from support via a blacklist
 configuration
  // The following section contains a whitlelist with supported App environments
  // All values must be strings or string arrays
 whiteList =
  [
  [
  AppID : "de.softpro.SignDocMobile", // The ID of the App
  AppVersion : "1.2.0", // The version of the App
  OS : "iPhone OS",
  OSVersion : "5.1.1",
  DeviceID : "42:00:00:00:00:E9", // CID/unique device identifier
  DeviceModel : "iPhone4,1", // e.g. "iPhone4,1", "iPad3,4" (iPhone 4S, iPad 4)
  DeviceVendor : "Apple",
  ScreenSize : "1024x768@2", // (width x height @ pixel density)
  CPUModel : "12.6", //  "12.10",... (ARM v6, v7, v7F, ...), according to iPhone SDK
 mach/machines.h
  CPUClock : "630000000", // speed in Hz
  CaptureType : [ "SPen", "CPen" ], // "SPen", "CPen" and "Mouse"
  ConfigLinkID : "000" // unique link ID generated by sdm_config
  ],
  [
  OS : "iPhone OS",
  DeviceModel : "iPhone4,1", // e.g. "iPhone4,1", "iPad3,4" (iPhone 4S, iPad 4)
  DeviceVendor : "QuApple",
  CaptureType : [ "SPen", "CPen" ] // "SPen", "CPen" and "Mouse"
  ]
 ]
 blackList =
 [
  [
  AppID : "de.softpro.SignDocMobile", // The ID of the App
```

```
  AppVersion : "1.2.0", // The version of the App
  OS : "iPhone OS",
  OSVersion : "5.1.1",
  DeviceID : "42:00:00:00:00:E9", // CID/unique device identifier
  DeviceModel : "iPhone4,1", // e.g. "iPhone4,1", "iPad3,4" (iPhone 4S, iPad 4)
  DeviceVendor : "Apple",
  ScreenSize : "1024x768@2", // (width x height @ pixel density)
  CPUModel : "12.6",
  CPUClock : "630000000", // speed in Hz
  CaptureType : [ "SPen", "CPen" ], // "SPen", "CPen" and "Mouse"
  ConfigLinkID : "000" // unique link ID generated by sdm_config
  ],
  [
  OS : "iPhone OS",
  DeviceModel : "iPhone4,1", // e.g. "iPhone4,1", "iPad3,4" (iPhone 4S, iPad 4)
  DeviceVendor : "Apple"//,
  msgid : "blacklist1"
  ],
  [
  AppID : "de.softpro.SignDocMobile", // The ID of the App
  AppVersion : "1.2.0", // The version of the App
  questionable : "true",
  msgid : "blacklist2"
  ],
  [
  AppID : "de.softpro.SignDocMobile", // The ID of the App
  AppVersion : "1.1.5", // The version of the App
  msgid : "blacklist3"
  ]
 ]
}
```

Click to sign

SignDoc Web normally is used in conjunction with a handwritten signature. With this feature, basic support for Click to Sign (C2S) is added to SignDoc Web core.

Description

SignDoc Web provides basic support for c2s images. The appearance of the signature field can be freely designed by the user. The basics of C2S consists basically of these building blocks:

- Signature Appearance Plugin (Interface: C2SSignatureRenderer)
  This plugin is responsible for rendering the visual appearance of a signed signature field.
- Server settings to start the SignDoc Web Server in "C2S mode" The C2S signing ceremony differs from a traditional signing ceremony. C2S does not need a browser extension to capture a handwritten signature.

Signing ceremony

Since there is no need to capture handwritten signatures in C2S environment, the user interface when signing differs quite a bit. Instead of writing a signature, the user enters his name in a dialog box when signing. This information is rendered into the visible signature appearance along with other information (for details see below).

Configuration options in sdweb_config.groovy

C2S settings

- **sdweb.c2s.mode.force** (boolean): If true, the server is running in "C2S mode". This means, that all signature fields are treated as C2S signature fields. Supported values: true, false. Default: false
- **sdweb.c2s.defaults.signaturerenderer.impl** (string): The Signature Renderer Plugin to use. The default Implementation uses an SVG Renderer to create the appearance of the signature field. Supported values: a valid PluginId. Default: 'de.softpro.sdweb.plugins.impl.SimpleC2SSVGRenderer'
- **sdweb.c2s.signaturereimage.dpi** (integer): DPI of the created signature image. Supported values: > 0. Default: 300

HTTP servlet parameters

- **c2s_uid** Is a free to use parameter that is supposed to carry any user identification information associated with the current document process.

  This information can be part of the AuditTrail. This information can be evaluated and read out by plugins
- **c2s_tid** Is a free to use parameter that is supposed to carry any transaction relevant information associated with the current document process.

  This information can be part of the AuditTrail. This information can be evaluated and read out by plugins.

Other useful settings

- **sdweb.workflow.skip_bp_detection** Setting the value to false disables the check for the browser plugin, that is executed whenever a document is opened. Supported values: true, false

Signature field appearance

If the default plugin (SimpleC2SSVGRenderer) is used, the default signature appearance can be found in the directory:

```
SDWEB_HOME/c2s/softpro_c2s_template.svg
```

The format of the file is SVG. The file serves as template for the single C2S signatures and the design can be adapted to meet specific needs. This svg file can contain some keywords, that will replaced with concrete values when signing.

The keywords are:

Keyword	Description
%%%C2S_VAR_TIMESTAMP%%%	The signing timestamp
%%%C2S_VAR_SIGNER_NAME%%%	The signer's name
%%%C2S_USER_ID%%%	The signer's ID e.g. email address
%%%C2S_IP_ADDRESS%%%	The IP address of the signer when applying the signature

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
```

```
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:ev="http://www.w3.org/2001/xml-events"
    version="1.1" baseProfile="full" width="300px" height="100px" viewBox="0 0 300
 100">
  <rect id="c2s_signature_background" x="0" y="0" width="300" height="100"
 style="fill:#ffffff" />
    <text id="c2s_fixed_esigned"   x="5"  y="12" font-size="12"
 style="fill:#000000;font-family:Verdana">E-signed: </text>
    <text id="c2s_var_signdate"    x="80" y="12" font-size="12"
 style="fill:#000000;font-family:Verdana">%%%C2S_VAR_TIMESTAMP%%%</text>
    <text id="c2s_var_signer_name" x="5"  y="45" font-size="36"
 style="fill:#0000ff;font-family:Brush Script MT">%%%C2S_VAR_SIGNER_NAME%%%</text>
    <text id="c2s_var_user_id"     x="5"  y="65" font-size="14"
 style="fill:#000000;font-family:Verdana">%%%C2S_USER_ID%%%</text>
    <text id="c2s_fixed_ipaddress" x="5"  y="80" font-size="12"
 style="fill:#000000;font-family:Verdana">IP:</text>
    <text id="c2s_var_ipaddress"   x="25" y="80" font-size="12"
 style="fill:#000000;font-family:Verdana">%%%C2S_IP_ADDRESS%%%</text>
  <rect id="c2s_electronic_signature_background" x="165" y="84" rx="5" ry="5"
 width="135" height="15" style="fill:#007700" />
  <text id="c2s_fixed_electronic_signature" x="175" y="95" font-size="10"
 style="fill:#ffffff;font-family:Verdana">Electronic Signature</text>
</svg>
```

Cluster environment

SignDoc Web depends on a server side session. To be able to work in a clustered environment, some configurations must be done.

SDWEB_HOME directory

Each instance of SignDoc Web requires a SDWEB_HOME directory. In a clustered environment all nodes should have the same SDWEB_HOME. This can generally be achieved by different strategies:

- Containerize the SignDoc Web instance (e.g. by using Docker) including SDWEB_HOME
- Put the SDWEB_HOME directory on a common network share (read/write access is required)
- Explicitly synchronize the SDWEB_HOME on all nodes with a synchronization tool like e.g. rsync

Session cookie / Sticky session / Session state persistence

If SignDoc Web is hosted behind a load balancer or reverse proxy it is required to configure it to use sticky sessions. This is achieved by respecting the applications servers session cookie.

Example for Apache Web Server: https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html

Replicating session state

To improve high availability, the server side session must be persisted on server side. This can be done in different ways. It is generally recommended to use a fast in-memory cache like memcached but traditional database are also possible. Whenever a node of a cluster fails, another node can pick up the sessions that existed on the failing node. The user of SignDoc Web will not notice this and can continue to work.

Example for Tomcat: https://github.com/magro/memcached-session-manager

Preloading documents

Preloading documents is a special functionality of SignDoc Web that means that documents (usually PDF documents) are pushed to SignDoc Web - on server side - before they can be opened on client side via a refid parameter. The refid parameter is usually a random UUID string. From a security point of view, this is a best practice. Preloading must be done, whenever the application producing the documents to sign is not a web application (e.g. a terminal application).

To be able to meet different technical requirements and/or strategies, the preload implementation logic is implemented via an IPreloadPlugin* plugin. There are currently 3 standard preload plugins delivered with SignDoc Web using different strategies. See Preload plugins for a detailed description.

Configuration of toolbar and GUI

SignDoc Web starts up with a default setting for the visibility of the toolbar icons. The below icons are available and can be enabled or disabled in the mobile_configuration.xml (Mobile GUI for Desktop and mobile devices) or signdoc_configuration.xml (Desktop GUI) file. Please note that mobile_configuration.xml contains two separate sections for toolbar configuration, one for touch devices and one for desktop browsers.

- **TA_FinalizeDocument** Sets the visibility of the toolbar icon **Send to Archive**. Supported values: true, false. Default: false
- **TA_Cancel** Sets the visibility of the toolbar icon **Cancel document**. Supported values: true, false. Default: true
- **TA_Download** Sets the visibility of the toolbar icon **Download Document**. Supported values: true, false. Default: true
- **TA_OpenDocument** (only available on Desktop if used together with Remote Interface). Sets the visibility of the toolbar icon **Open new document**. Supported values: true, false. Default: false
- **TA_Email** (only available on Desktop if used together with Remote Interface). Sets the visibility of the toolbar icon **E-Mail document**. Supported values: true, false. Default: false
- **TA_Print** Sets the visibility of the toolbar icon **Print document**. Supported values: true, false. Default: true
- **TA_FirstPage** Sets the visibility of the toolbar icon **First page**. Supported values: true, false. Default: true
- **TA_LastPage** Sets the visibility of the toolbar icon **Last page**. Supported values: true, false. Default: true
- **TA_NextPage** Sets the visibility of the toolbar icon **Next page**. Supported values: true, false. Default: true
- **TA_PrevPage** Sets the visibility of the toolbar icon **Previous page**. Supported values: true, false. Default: true
- **TA_PageInfo** (only available in Mobile GUI). Sets the visibility of the toolbar icon **Page Info**. Supported values: true, false. Default: true
- **TA_PageInput** (not available for mobile-view). Sets the visibility of the toolbar icon **Select a page** within the document. Supported values: true, false. Default: not set
- **TA_Highlight** Sets the visibility of the toolbar icon **Highlight input fields**. Supported values: true, false. Default: true
- **TA_ZoomList** (not available for mobile-view). Sets the visibility of the toolbar icon **Select zoom** in %. Supported values: true, false. Default: true

- **TA_ZoomIn** Sets the visibility of the toolbar icon **Zoom in** the document. Supported values: true, false. Default: true
- **TA_ZoomOut** Sets the visibility of the toolbar icon **Zoom out** the document. Supported values: true, false. Default: true
- **TA_ZoomToWidth** (only available in Mobile GUI). Sets the visibility of the toolbar icon **Zoom to width** of the document. Supported values: true, false. Default: false
- **TA_About** (not available for mobile-view). Sets the visibility of the toolbar icon **About page**. Supported values: true, false. Default: true
- **TA_ClearSignature** Sets the visibility of the toolbar icon **Clear Signature**. This is only supported in toolbar configuration section for touch devices. Supported values: true, false. Default: false
- **TA_DisplayFields** Sets the visibility of the toolbar icon **Display fields dialogue**. Supported values: true, false. Default: true
- **TA_AddCaptureField** Sets the visibility of the toolbar icon **Add a capture field**. Supported values: true, false. Default: false
- **TA_AuditTrail** Sets the visibility of the toolbar icon **Audit Trail**. Supported values: true, false. Default: false
- **TA_Help** Sets the visibility of the toolbar icon **Help**. Supported values: true, false. Default: true

See further GUI or workflow configuration options in the files mobile_configuration.xml (Mobile GUI for desktop and mobile devices) or signdoc_configuration.xml (Desktop GUI).

Configure server

sdweb_config.groovy is the main configuration file for the SignDoc Web server.

It is located at:

```
%SDWEB_HOME%/conf
```

The general syntax is:

```
a.config.key = <JAVA object>
```

Examples

```
integer.config.key = 123 // no quotes!
string.config.key = "John Doe" // quotes are mandatory
boolean.config.key = true // no quotes!
```

This example shows the general syntax of the configuration file. It is based on Groovy config slurper file syntax, but written as traditional Java properties files.

In contrast to traditional Java properties files the values are Java Objects - not only Strings

When the Plugin configuration is injected to the plugins, the value of a key can be accessed in this way:

```
void injectPluginConfiguration(Map<String,Object> configMap) {
  int intValue = (Integer)configMap.get("integer.config.key")
  String strValue = (String)configMap.get("string.config.key")
  boolean boolValue = (Boolean)configMap.get("boolean.config.key")
```

```
}
```

Put string values in quotes like in Java programs. In traditional Java Properties files you must not use quotes.

To quote the string values you can either use double quotes ("StringValue") or single quotes ('StringValue').

When a node contains a dot (.) character you have to put the node name (see example above) in quotes or create a subnode with every dot.

Use forward slashes whenever a directory path is required.

If you want to use ${sdweb.home} variable (path to sdweb_home directory) in the configuration

(e.g. sdweb.signature.watermark.image.template_dir="${sdweb.home}/wm")

you have to set the following statements in sdweb_config.groovy before:

```
import de.softpro.sdweb.SdWebHelper

sdweb.home = SdWebHelper.SDWEB_HOME;
```

Further information on configuration parameters can be found in the Configuration file sdweb_config.groovy.

Description of sdweb.browserplugin.padclass

A cable bounded pad is selected first by default if connected. If no pad is connected, the Sign2Phone pairing dialog is offered (if Sign2Phone is installed).

The padclass entry can contain one or more tablet class entries (separated by semicolons) which determine the search order for capture devices.

Example

If you want to use Sign2Phone irrespective of any other connected sign pad devices you can enter

```
sdweb.browserplugin.padclass="SPTabletRemoteTablet"
```

Valid tablet classes are:

- SPTabletWSignPad interface to Wacom SignPad tablets
- SPTabletWTablet interface to Kofax eInk driver to access TabletPCs
- SPTabletRemoteTablet interface to Kofax driver to access Smartphones as tablets
- SPTabletHidDrv interface to the Kofax HID driver to access TabletPCs
- SPTabletInterlink interface to the Kofax Interlink driver (SWILUniv)
- SPTabletMobinetix interface to Mobinetix driver
- SPTabletStepOver interface to StepOver BlueM and BlueM-LCD tablets
- SPTabletTopaz interface to Topaz 1X5, 4X3 and 4X5 SE LCD tablets
- SPTabletWacom interface to Wacom Intuos, Graphire, etc. tablets
- SPTabletVerifone interface to Kofax driver to access Verifone Mx 800 series tablets

- SPTabletGeneric interface to Kofax driver to access drivers that implement the documented tablet access

If the clients have the possibility to capture a signature with different devices you can define the tablet search order with the padclass definition in sdweb_config.groovy.

Example

The client has a sign pad (e.g. STU-530) but also Sign2Phone installed. If the user wants to have the possibility to capture signatures with both devices (iPhone and STU-530) with Sign2Phone having a higher priority the administrator must define:

```
sdweb.browserplugin.padclass="SPTabletRemoteTablet;SPTabletWSignPad"
```

The SPTabletRemoteTablet as primary entry causes the Sign2Phone pairing dialog to be displayed. If the user doesn't want to sign with his iPhone he can switch to the STU-530 (second tablet class definition SPTabletWSignPad) capturing by cancelling the pairing dialog.

This does not work in the same way the other way round. If the user wants to use two devices with priority on the connected pad he can not switch to the Sign2Phone by clicking on the Cancel button in the capture dialog. If he want to use Sign2Phone (instead of the connected pad) the user has to disconnect the pad first. For this case the administrator doesn't need to configure anything, because it is the default behavior.

Note The padclass configuration in sdweb_config.groovy is not considered if FirstWT=drv is defined in local tablet.ini, for example FirstWT=SP_WspDrv.

Description of sdweb.browserplugin.padconfiguration

The sign pad model is referenced by a specific padclass (parameter sdweb.browserplugin.padclass).

Additional configuration can be performed via this parameter on the below padclassed:

- SPTabletWSignPad interface to Wacom SignPad tablets
- SPTabletRemoteTablet interface to Kofax driver to access smartphones as tablets
- SPTabletStepOver interface to StepOver BlueM and BlueM-LCD tablets
- SPTabletTopaz interface to Topaz 1X5, 4X3 and 4X5 SE LCD tablets
- SPTabletVerifone interface to Kofax driver to access Verifone Mx 800 series tablets

Example

If you want to specify a fixed language in the Sign2Phone pairing dialog for all clients you could define:

```
sdweb.browserplugin.padconfiguration="[REMOTETABLET]\nLanguage=en\n"
```

Custom CSS configuration

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation semantics (the look and formatting) of a document written in a markup language and also for SignDoc Web.

It is possible to configure one or more custom specific CSS files in sdweb_config.groovy.

For browser desktop layout changes you can define a list of CSS files with:

```
sdweb.ria.signdoc.css_list_custom=["SignDocCustom1.css", "SignDocCustom2.css"]
```

The CSS files can be defined:

for iOS mobile devices

```
sdweb.ria.signdoc_mobile.ios.css_list_custom=["SignDocMobileCustom_IPad_1.css"]
```

and for Android

```
sdweb.ria.signdoc_mobile.android.css_list_custom=["SignDocMobileCustom_Android_1.css",
"SignDocMobileCustom_Android_2.css"]
```

and all other mobile devices

```
sdweb.ria.signdoc_mobile.css_list_custom=["SignDocMobileCustom.css"]
```

It is also possible to define an additional CSS file in the document (loadby...) request with the style=.... parameter.

Example

```
style=DocSpecific.css
```

These CSS definitions have higher priority than other previously defined layout settings.

These customer specific CSS files are expected to be outside of the sdweb package in:

```
%SDWEB_HOME%/css
```

The (http) load request for a SignDoc Web document can contain some general parameters, like the docid, the dmsid or the result url.

In addition to those parameters it is possible to define here also the 'style' parameter with a customer specific CSS file name as value.

This CSS file must be available in the

```
%SDWEB_HOME%/css
```

directory. If defined it overwrites any other defined CSS settings (insofar there is an overlapping).

Example

http://localhost:8080/sdweb/load/bytemplate?
template=account_opening.pdf&docid=MyDoc&dmsid=de.softpro.sdweb.plugins.impl.FileDms&style=DocSpecific.css

Note Note This setting is only valid for the current document.

SignDoc Web is enabled to address image files from:

```
%SDWEB_HOME%/resources
```

External images from

```
%SDWEB_HOME%/resources
```

can be referenced if the following setting in sdweb_config.groovy is set to true:

```
sdweb.ria.image.custom.enabled=true
```

By default custom image handling is not enabled!

In order to address a custom specific image file you must define it in the context prefix custom/img.

Example usage:

You can define your own CSS file with:

```
sdweb.ria.signdoc.css_list_custom=["SignDocCustom1.css"]
```

This CSS file (located in %SDEB_HOME%css) could include the following style information, in order to replace the print icon with you own image (myPrint.gif):

```
.icon-toolbar-print {
background: url("custom/img/myPrint.gif") no-repeat center center !important;;
}
```

The default CSS files can be used as blueprint for customized CSS files.

For the desktop gui the default CSS file is:

```
WEB_SERVER\webapps\sdweb\gwt\de.softpro.sdweb.gwt.SignDoc\SignDoc.css
```

For the mobile gui the default CSS files are:

for android devices

```
WEB_SERVER\webapps\sdweb\gwt\de.softpro.sdweb.gwt.SignDocMobile
\SignDocMobile_Android_1.css
```

for iPad devices

```
WEB_SERVER\webapps\sdweb\gwt\de.softpro.sdweb.gwt.SignDocMobile
\SignDocMobile_IPad_1.css
```

and for other devices

```
WEB_SERVER\webapps\sdweb\gwt\de.softpro.sdweb.gwt.SignDocMobile
\SignDocMobile.css
```

Digital signature certificate

Description

SignDoc Web can make use of a PKCS#12 certificate store to use the embedded certificate for signing each signature. This makes it possible to use trusted (not self signed) certificates to sign a digital signature field, so that e.g. Adobe Reader can validate the certificate used to digitally sign the signature field.

By default, SignDoc Web uses a Demo Certificate, that should be replaced by the user. If SignDoc Web should not use a custom certificate, the SignDoc Web administrator should delete the demo certificate store: SDWEB_HOME/conf/cert_store.p12

Note Note Additional documentation can be found in file SDWEB_HOME/conf/cert_store_readme.txt.

Prerequisites

The certificate must be in PKCS#12 format

The PKCS#12 file must contain only 1 certificate

Useful tools

Portecle: Create PKCS#12 certificate stores

Configuration

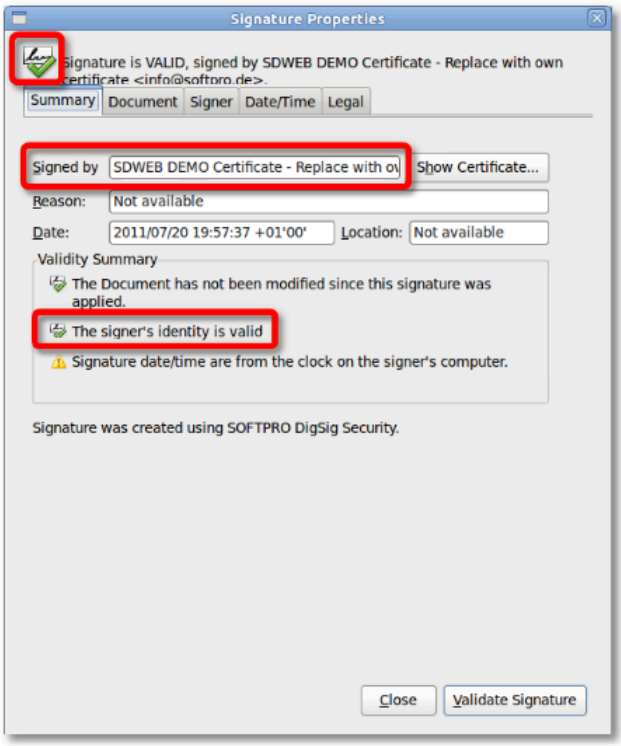
```
// location of the PKCS#12 certificate store
sdweb.certificate.store.pkcs12.file="c:/sdweb_home/conf/cert_store.p12"
// the password of the PKCS#12 certificate store
sdweb.certificate.store.pkcs12.password="secret"
```

Usage

Adjust the configuration options to use the certificate. The PKCS#12 file must be readable by the SignDoc Web Server process. If the configuration setting does not point to a valid PKCS#12 file, the server will use the standard self-signed "one time" certificates.

Note Note When the Adobe Reader validates the signatures certificate, it needs to trust the certificate to produce a green tick.

Verification in Adobe Reader



Dynamic tablet screens

Description

Creating and displaying dynamic tablet layouts for signpads depending on the document content and the language.

What can be displayed/hidden?

- Company logo/name
- Date
- Declaration of agreement
- Disclaimer
- Name of the signer
- Account number, amount (e.g. cash withdrawal)

The files WacomSTUSeries.xml and TabletScreenLayout.xsd are defining the graphical display of tablet layouts and are stored in the installation directory:

```
SDWEB_HOME/tablet_screens/
```

The WacomSTUSeries.xml file contains the default definition of the SOFTPRO dynamic layouts for STU-300, STU-500, STU-430, STU-520, STU-530, DTU-1031, DTU-1631 and Tablet PCs.

The TabletScreenLayout.xsd file describes an XML scheme including documentation. It defines the rules for creating the dynamic layout.

Note The default coordinate system uses relative coordinates. The dimension of each tablet layout has 1000 units in width and height.

The default coordinate system uses relative coordinates. The dimension of each tablet layout has 1000 units in width and height.

Usage

To be able to use custom dynamic signature screens, 2 basic options exist.

- Default layout

  The XML document having an XML element <tns:LayoutId> set to the value default will be used for all signing ceremonies unless a different tablet screen is defined for a specific signature field.

  See SDWEB_HOME/tablet_screens/WacomSTUSeries.-xml
- Custom layout for inserted signature fields

  Add the attribute

  ```
  screenlayout=<LayoutID>
  ```

  to the cmd statement inserting a new signature field. When signing this signature field, the specified layout will be used.

  Example

  ```
  name=sig1|page=1|type=formfield|subtype=signature|bottom=10|left=10|
  width=150|height=50|screenlayout=piggybank_example
  ```

  See SDWEB_HOME/tablet_screens/piggybank_example.xml

Description of XML elements

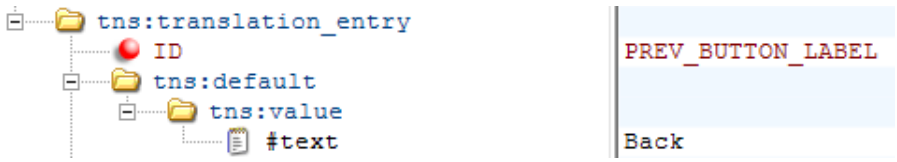
`SDWEB_Home/tablet_screens/WacomSTUSeries.xml`

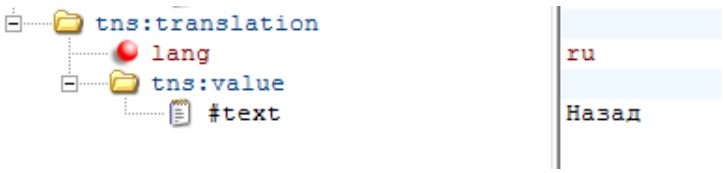
Element	Description
ActionButton	The element is used to submit user actions. The action is defined by content of the sub element <ActionId>. The background-color of action buttons is set to transparent for STU-500 and STU-300 devices.
DefaultFont	The element defines the default font of the particular canvas.
Image	An image can be defined inline base64 encoded or via external URL reference.
TextBox	The element consists of text, which is displayed as continuous text. The text will be automatically wrapped, when the text line exceeds the defined width of the element.
TextLine	The element consists of text, which is displayed in one line. The text will not be automatically wrapped. A suitable font size is automatically chosen, so that the text is adapted to the defined rectangle. The default font size can be overridden by a user-defined font size.
Rectangle	The element is a rectangle. As an example for the rectangle of a height=0 is a line below the signature.

Note The language of the button's label can be changed in the TextTranslationTable.xml that is located in the directory sdweb_home/i18n by adding a new value to the translation_entry element.

Example

Adding translation to the label of action button **Back**





Description of the attributes

Attribute Description	ActionButton	DefaultFont	Image	TextBox	TextLine	Rectangle
background-color Defines the color of the rectangle's enclosing area.	x		x	x	x	x
blackAndWhiteDithering Defines dithering type for the image when reducing the colors to black and white			x			
border-color Defines the color of the rectangle's border line	x		x	x	x	x
border-width Line width of the border in pixel.	x		x	x	x	x
default-z-order The default z-order of the element. 0 means background.	x		x	x	x	x
font-size Defines dithering type for the image when reducing the colors to black and white						
font-weight either REGULAR or BOLD						

Attribute Description	ActionButton	DefaultFont	Image	TextBox	TextLine	Rectangle
element_id The element id of a layout element. This id can be used to reference the element within a SPDynamicContentMap element.	x		x	x	x	x
having_round_corners Defines, if the corners of a rectangle are rounded.	x		x	x	x	x
height The height of an element	x		x	x	x	x
left The left coordinate of an element.	x		x	x	x	x
origin The origin of the elements coordinate system.	x		x	x	x	x
text-color Color of the text.	x			x	x	
text-halign Horizontal alignment of the text.	x			x	x	
text-valign Vertical alignment of the text.				x	x	
top The top coordinate of an element.	x		x	x	x	x
unit Defines the format of coordinates.	x		x	x	x	x
user-z-order The user-defined z-order of the element. 0 means background. The default-order is defined by attribute default-z-order (fixed value) and is used if 2 elements have the same z-order.	x		x	x	x	x
width The width of an element.	x		x	x	x	x
word-wrapping Defines if word-wrapping should be done.				x		

Encrypt sensitive data in sdweb_config.groovy configuration file

Sometimes it makes sense to encrypt data in the sdweb_config.groovy config file. This is especially useful for passwords.

All String config entries can be stored encrypted in the sdweb_config.groovy file. To encrypt the data use the PasswordEncryptionHelper tool that is located in directory:

```
SDWEB_HOME/tools
```

The file name is:

```
PasswordEncryptionHelper_<SignDoc Web Version>.jar
```

To make the encrypted values available in the config file, the "real setting key" has to be modified and appended with the suffix ".encrypted_string". Whenever the server finds an encrypted value, it will use the decrypted value as the value for the "real setting key".

Usage

```
java -jar PasswordEncryptionHelper <key-size:128|192|256> [password to
encrypt]
```

Example

Store the PKCS#12 password encrypted in the configuration.

Create the encrypted value

Encrypt the string "secret" using 256 bit strength so it can be used in the SignDoc Web configuration file.

Command:

```
java -jar PasswordEncryptionHelper_4.1.jar 256 secret
```

Output:

```
PasswordEncryptionHelper version: $Name: RST#SignDocWeb#core#4-1-090 $
Encrypted Password (key-size=256): c941a5d5fddb22e067752e8741cf251b
Decrypted Password: secret
```

Use the encrypted value

Add the following lines to sdweb_config.groovy:

```
// set the encryption strength in bits
sdweb.config.encrypt.strength=256
// set the encrypted password (in one line!)
sdweb.certificate.store.pkcs12.password.encrypted_string="c941a5d5fddb22e067752e8741cf251b"
```

In order to use an encryption key length > 128 bits the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files are needed. The files are currently available at the Java SE download page. This zip file contains a couple of policy jars, which you need to copy over the top of the ones already in the {java.home}/jre/lib/security directory of your JRE.

External help

Some customers want to create their own Help and host it separately. The Help must be available via URL Link.

The URI to customer specific help can be specified in sdweb_config.groovy with sdweb.custom.help.

Example

```
sdweb.custom.help="http://customer:8080/sdweb_help"
```

The language code of the requesting browser client is appended to this URL with:

```
lang=...
```

Example

```
sdweb.custom.help="http://customer:8080/sdweb_help?lang=de"
```

Note For desktop browsers sdweb.custom.help points to the default help URI if nothing is defined.

Font mapping configuration

The font mapping configuration is done in the file:

```
SDWEB_HOME/fonts/SPFontConfig.xml
```

This defines a mapping between font files for usage in SignDoc Web documents. This file can be edited and adjusted.

SPFontConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SPFontConfig SYSTEM "SPFontConfig.dtd">
<SPFontConfig>
  <!-- Example: Windows Standard Font Directory -->
  <!--<FontFiles>c:\Windows\Fonts\verdana*.ttf</FontFiles>-->
  <FontFiles>%%%FONT_FILES_DIRECTORY%%%\DejaVuSerif.ttf</FontFiles>
  <FontFiles>%%%FONT_FILES_DIRECTORY%%%\DejaVuSans.ttf</FontFiles>
  <Substitution>
    <FontName>SignDoc Standard</FontName>
    <FontName>DejaVu Sans</FontName>
  </Substitution>
  <Substitution>
    <FontName>Times Roman</FontName>
    <FontName>DejaVu Serif</FontName>
  </Substitution>
  <Substitution>
    <FontName>Helv</FontName>
    <FontName>DejaVu Sans</FontName>
  </Substitution>
  <Substitution>
    <FontName>Helvetica</FontName>
```

```
    <FontName>DejaVu Sans</FontName>
  </Substitution>
  <Substitution>
     <FontName>Arial</FontName>
     <FontName>DejaVu Sans</FontName>
  </Substitution>
  <Substitution>
    <FontName>TiRo</FontName>
    <FontName>DejaVu Serif</FontName>
  </Substitution>
</SPFontConfig>
```

SPFontConfig.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A font configuration file specifies font files and font substitutions. -->
<!ELEMENT SPFontConfig (FontFiles*, Substitution*)>

<!-- Contents: either the pathname of a single font or a pathname containing wildcard
 characters * and ? in the last component such as c:\fonts\*.ttf. Relative pathnames
 are relative to the directory containing the font configuration file. -->
<!ELEMENT FontFiles (#PCDATA)>

<!-- Defines one font substitution. Font substitutions are tried one by one until a
 match or the end is reached. There is only one level of font substitutions, that is,
 substituted names are not subject to substitution. Substitute the first font with the
 second one. Font names are case-sensitive. -->
<!ELEMENT Substitution (FontName,FontName)>

<!-- Contents: the font family name or PostScript name of the font to be substituted or
 of the font used as substitute. -->
<!ELEMENT FontName (#PCData)>
```

Gestures on mobile devices

Support of gestures on mobile dDevices

The basic gestures Swipe (change page) and PinchSpread (zoom) are implemented.

PinchSpread can be done with two fingers and is available on iPad, Android and Windows devices.

1-finger Swipe can be done with one finger on iPad and Android > 4.0 devices.

3-finger Swipe can be done on Windows and Android < 4.1 devices (1-finger swipe is technically not possible on these devices). For compatibility reasons a 3-finger Swipe can also be done on iPad and Android > 4.0 devices.

To differ a swipe gesture from a scrolling gesture the swipe gesture has these characteristics:

- If image width is larger than the screen width, the current scroll position has to be either 0 or the maximum possible position (this characteristic is only available for 1-finger swipe)
- The duration of the gesture is below a configured threshold
- The pixel distance between the gesture start and the gesture end position is above a configured threshold factor

Per default PinchSpread gestures and Swipe gestures are disabled via configuration.

PinchSpread configuration in mobile_configuration.xml

PinchSpread is enabled by adding it to the zoom list and setting it as default:

```
<component id="Lists">
.....
<element id="ZoomList">
<parameter key="Keys" value="PINCH_SPREAD"/>
<parameter key="Default" value="PINCH_SPREAD"/>
</element>
......
<component id="Miscellaneous">
.....
<element id="Zoomfactor">
<!--
The maximal allowed zoomfactor as provided in sdweb_config.groovy (sdweb.gui.zoom.max,
 default=300).
NOTE: should only be changed by Softpro!
-->
<parameter key="Max.Value" value="300"/>
<parameter key="Min.Value" value="75"/>
<!-- This factor is multiplied with the initial zoomfactor (1.5 * 100% = 150%) when
 pages are requested.
A higher value results in a better image quality when scaling.
This setting is only used if the default zoomlist entry is set to PINCH_SPREAD.
-->
<parameter key="PinchSpread.Factor" value="1.5"/>
</element>
```

Swipe configuration in mobile_configuration.xml

```
<component id="Miscellaneous">
.....
<element id="Swipe.Gesture">
<!-- Specify if the swipe gesture for changing pages should be enabled. -->
<parameter key="Enabled" value="false"/>
<!-- The maximum duration in milliseconds between touch start and touch end to identify
 a swipe event
when using 1-finger swipe (IPad, Android > 4.0). -->
<parameter key="Duration.Threshold" value="600"/>
<!-- The maximum duration in milliseconds between touch start and touch end to identify
 a swipe event
when using 3-finger swipe (Windows, Android < 4.1). -->
<parameter key="Duration.Threshold.3Finger" value="600"/>
<!-- The minimum distance factor between touch start and touch end event to identify a
 swipe event when
using 1-finger swipe (IPad, Android > 4.0).
The specified factor is multiplied with the available screen width to get the minimal
 distance in pixel which
is needed to identify the swipe gesture. If the current image width is less than the
 screen width the specified
factor is multiplied with the current image width.-->
<parameter key="Distance.Threshold.Factor" value="0.1"/>
<!-- The minimum distance factor between touch start and touch end event to identify a
 swipe event when using
3-finger swipe (Windows, Android < 4.1).
The specified factor is multiplied with the available screen width to get the minimal
 distance in pixel which
is needed to identify the swipe gesture. If the current image width is less than the
 screen width the specified
factor is multiplied with the current image width.-->
<parameter key="Distance.Threshold.Factor.3Finger" value="0.1"/>
</element>
```

PinchSpread is only possible if the current touch position is not in conflict with editable fields.

Swipe and PinchSpread gestures are also available when adding new capture fields.

The default zoom factor of PinchSpread is FIT_TO_WIDTH. This default is always used when:

- First page is initially displayed
- Page is changed
- Pages are pre-fetched
- Screen is rotated

Scaling the page via PinchSpread normally doesn't reload the page from the server.

The page is only reloaded if the user changes a field and scales the page afterwards (therefore the actual user changes are not visible in the scale view).

When PinchSpread is enabled the toolbar actions ZoomIn and ZoomOut can't be used any more (they are disabled) and should be removed from toolbar configuration.

Toolbar action

A new toolbar action FIT_TO_WIDTH has been added. This action can be used to reset the current zoom factor to FIT_TO_WIDTH. It is not visible per default.

Toolbar configuration in mobile_configuration.xml

```
<component id="Toolbar">
...
<element id="TA_ZoomToWidth">
<parameter key="Visible" value="false"/>
<parameter key="Tooltip" propertyFile="language"
 propertyKey="Toolbar.TA_ZoomToWidth.Tooltip"/>
<parameter key="Label" propertyFile="language"
 propertyKey="Toolbar.TA_ZoomToWidth.Label"/>
<parameter key="Description" propertyFile="language"
 propertyKey="Toolbar.TA_ZoomToWidth.Mobile.Description"/>
<parameter key="Description.Android" propertyFile="language"
 propertyKey="Toolbar.TA_ZoomToWidth.Mobile.Description"/>
</element>
```

Handling self-signed SSL certificates using iPads

iPhone configuration program

One way to provide configuration profiles for iPads is the use of the program iPhone Configuration Utility.

The iPhone Configuration Utility lets you easily:

- Create, update, encrypt and install configuration profiles
- Check for and install provisioning profiles and authorized applications
- Retrieve device information including console logs

Configuration profiles are XML files that contain the following information:

- Device security policies
- VPN configuration information
- Wi-Fi settings
- APN settings
- Exchange Account settings
- Mail settings and certificates

that allow the work with the iPhones and iPads in your corporate systems.

Download and installation

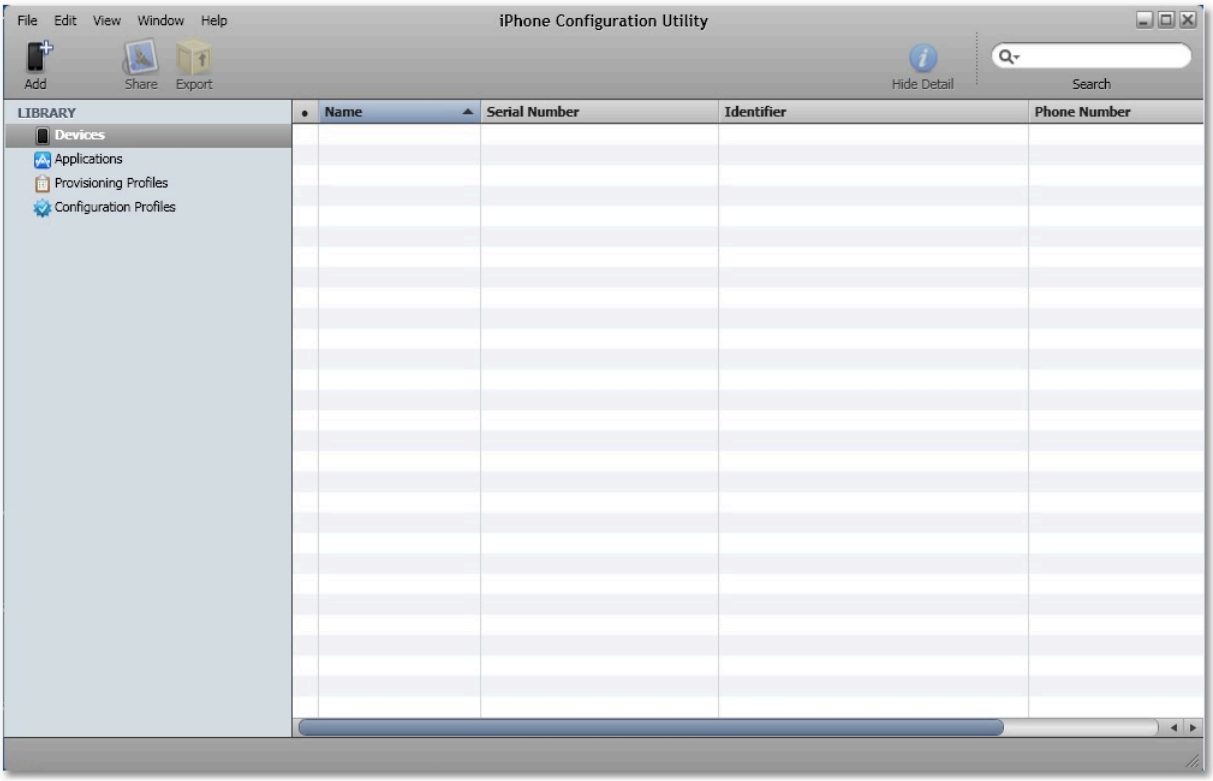
The iPhone Configuration Utility can be downloaded from the following website:

http://support.apple.com/kb/DL1466?viewlocale=en_US

A file named iPhoneConfigUtilitySetup.exe is stored on your computer. Start the installation by double clicking the EXE file. Follow the steps in the installation wizard.

Create configuration profile

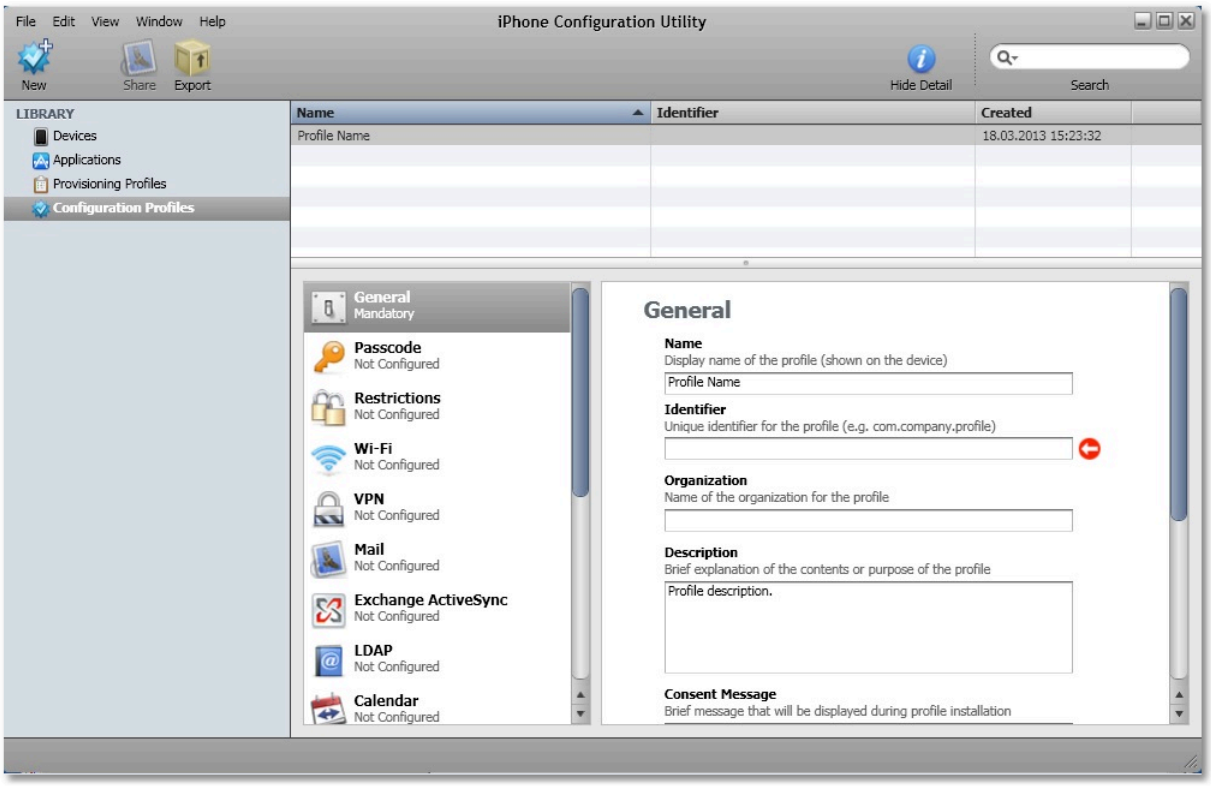
Launch the iPhone Configuration Utility. The home screen looks like this:



To create a new configuration profile, select the menu:

File > New Configuration Profile

A new configuration profile is created.



You can now change the general information for the configuration profile according to the requirements of your company system. In our example, we change the profile name to SignDocMobile and the identifier to de.softpro.SignDocMobile.

Thus, you created an empty configuration profile and can now add additional configuration information by selecting other categories in the list, such as WLAN and VPN settings.

Provide certificates

In this chapter we look at the provision of certificates, this is important in the case when you want to use your mobile device in a corporate environment with SSL encryption.

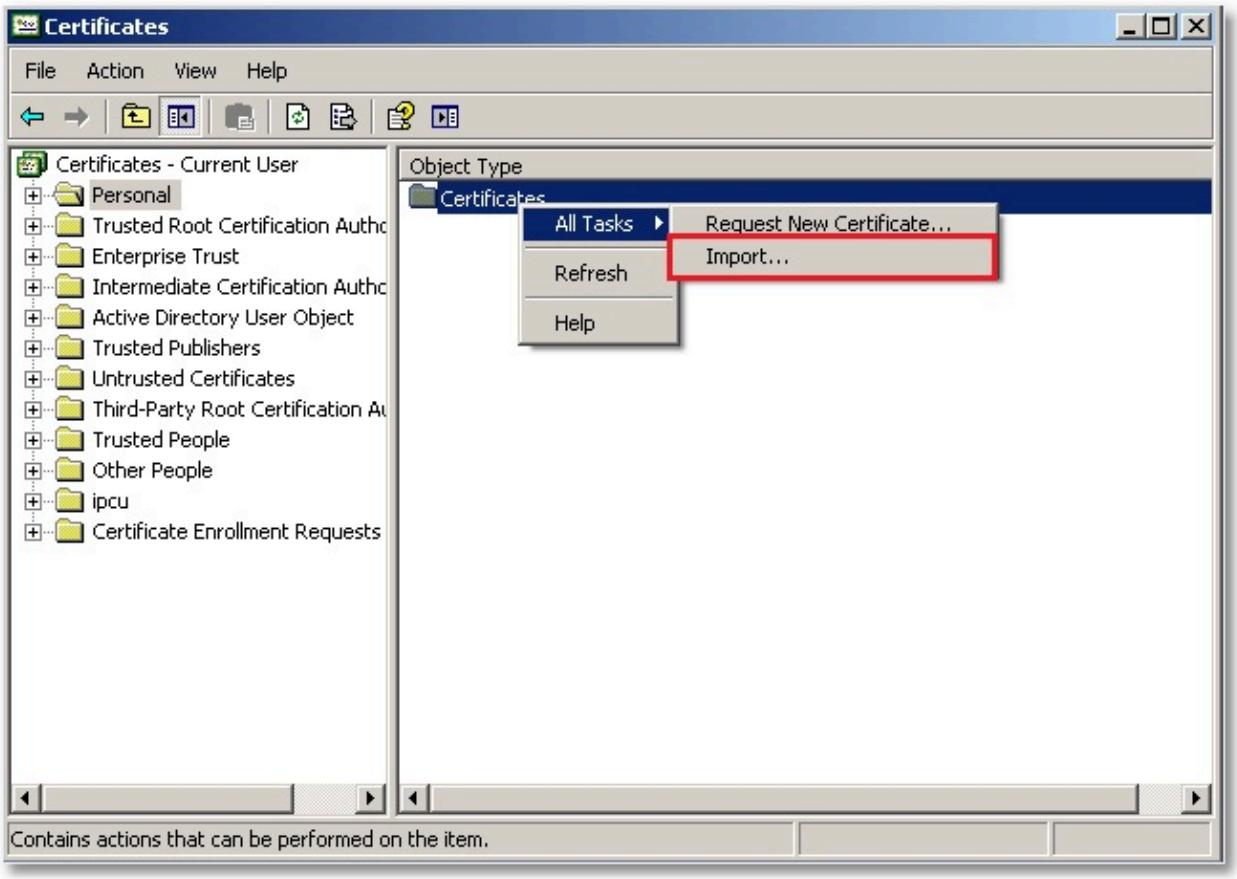
The iPhone Configuration Utility can use the already imported certificates that are installed in your Windows Certificate Manager. Below is a guide on how you can import your certificates in the Windows Certificate Manager.

iOS devices can use X.509 certificates with RSA keys. Here the file extensions .cer, .crt and .der are recognized. If the certificate is not available in your personal certificate deposit or key ring, you need to add it first.

Open the Windows Certificate Manager by clicking:

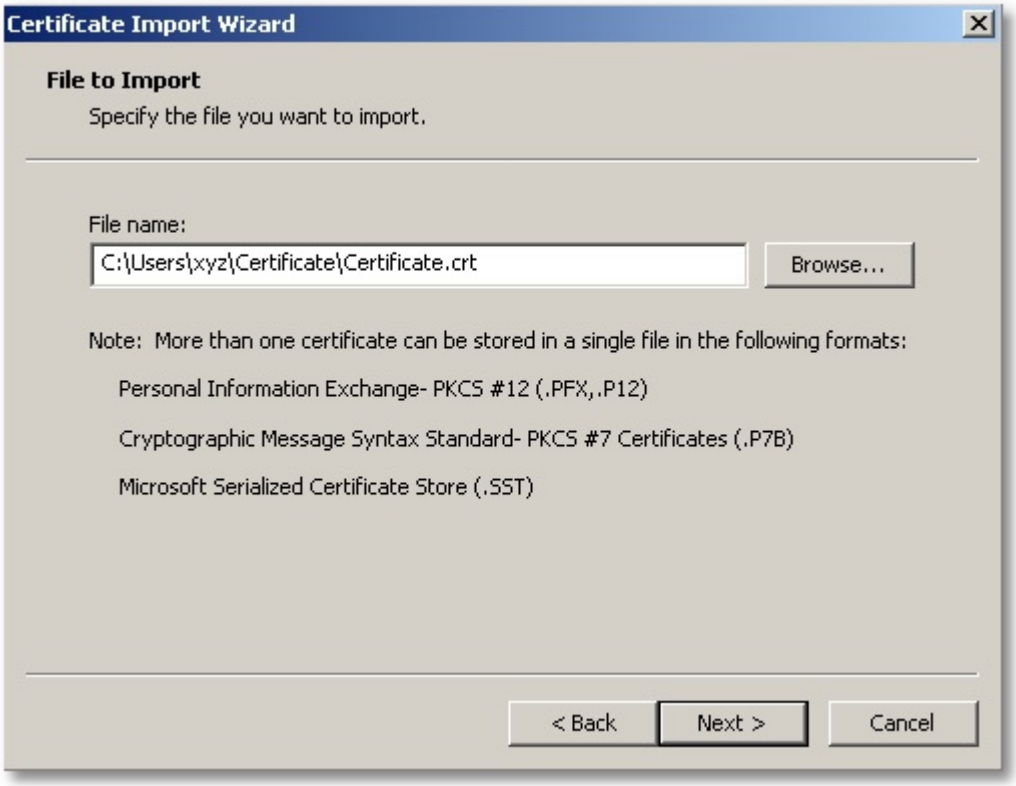
Windows Start Button > Control Panel > Credential Manager > Add a Certificate-Based Credential > Open the Certificate Manager

The Certificate Management page appears.

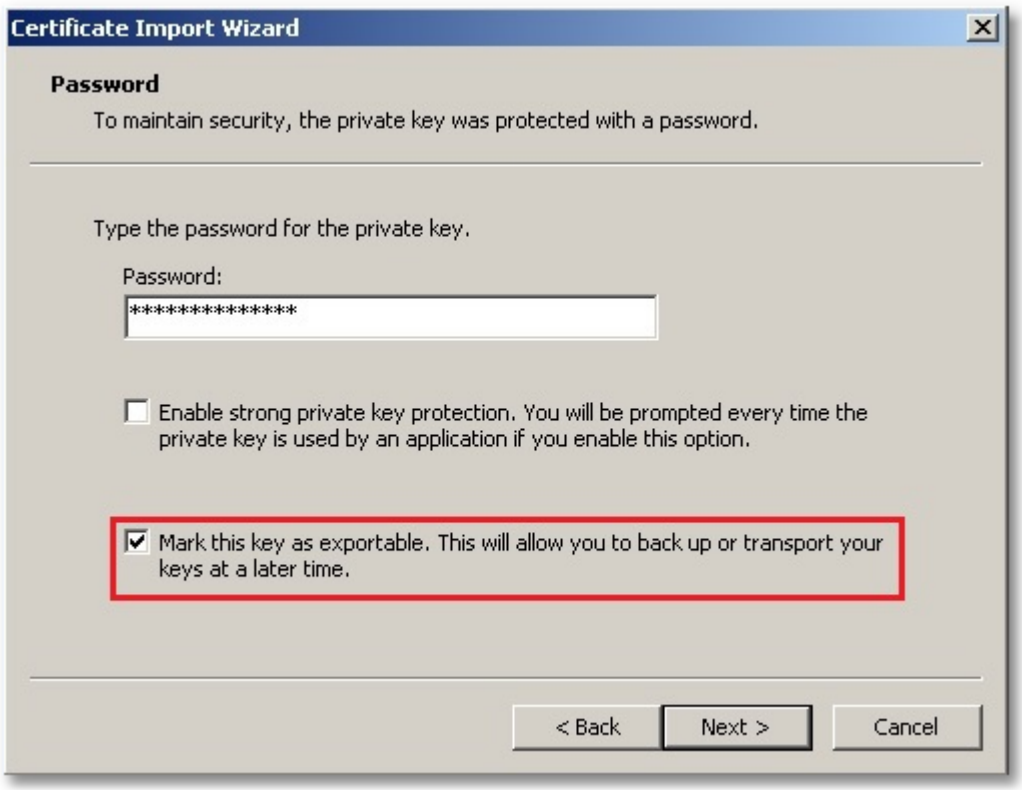


To import a certificate, expand the entry Personal. Right click on the Certificates directory and select Import from the menu.

The Certificate Import Wizard appears. In the Welcome dialog click on the Next button.



In the next dialog, specify the certificate file to import and click Next.



In the Certificate Import Wizard, you must specify the password for the certificate to be imported. Similarly, the private key must be marked as "exportable". This is one of the options of that are available from the wizard for importing certificates available.

In the next dialog box, select the certificate store. Be sure to select the radio button: Save all certificates in the following store.

and fill in the certificate store Personal. Click Next.

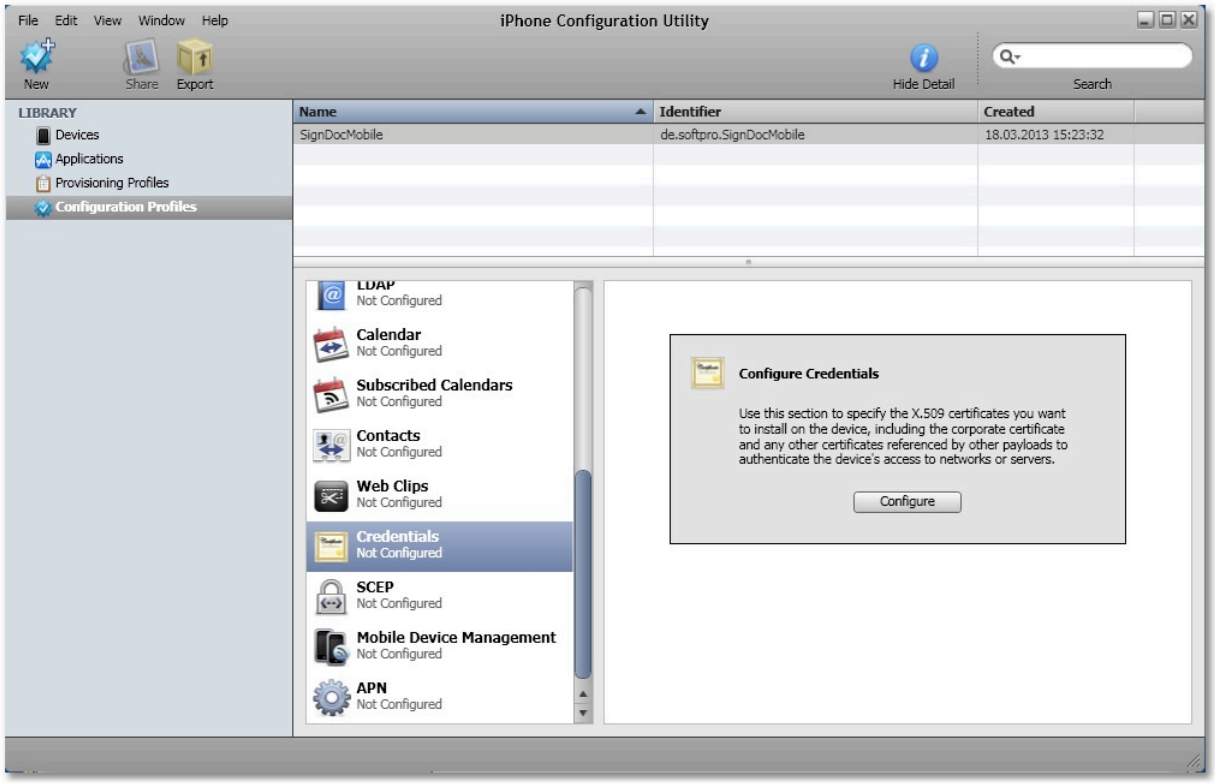
In the next dialog box, click Finish to import the certificate. If the import was successful, yet a success message is issued:

The import was successful.

In the dialog box, click OK. The imported certificate appears in the certificate management with its properties.

Add certificates to the configuration profile

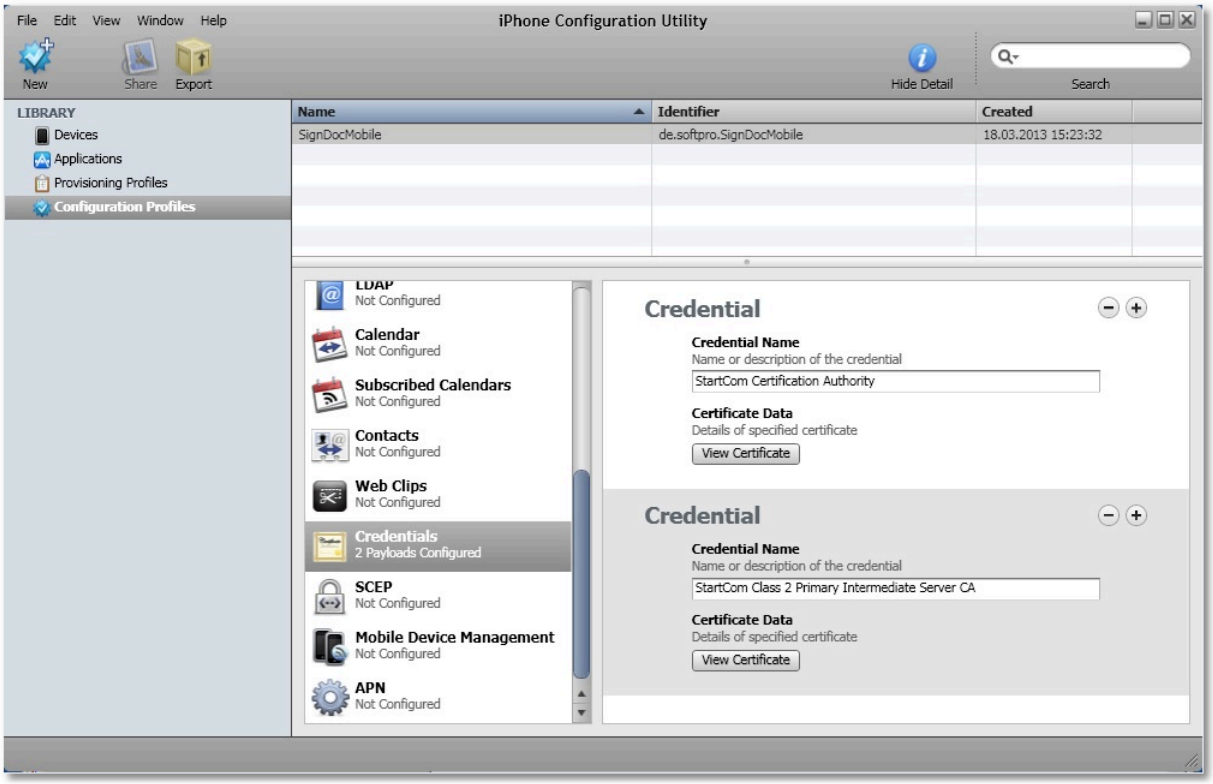
To add a certificate to the configuration profile, enter your library in the iPhone Configuration Utility and click on Configuration Profiles. Choose your configuration profile, from which you want to add a certificate. Scroll the list of properties you want to configure to the bottom and click on Credentials.



To configure a certificate, click the Configure button.

This opens a list with all the certificates that can be selected. This list contains also the imported certificates from the Windows certificate management. Select one and click OK.

The selected certificate is shown in the right panel. If you still want to add more certificates to a configuration profile, click the plus sign and the list of certificates is displayed again.



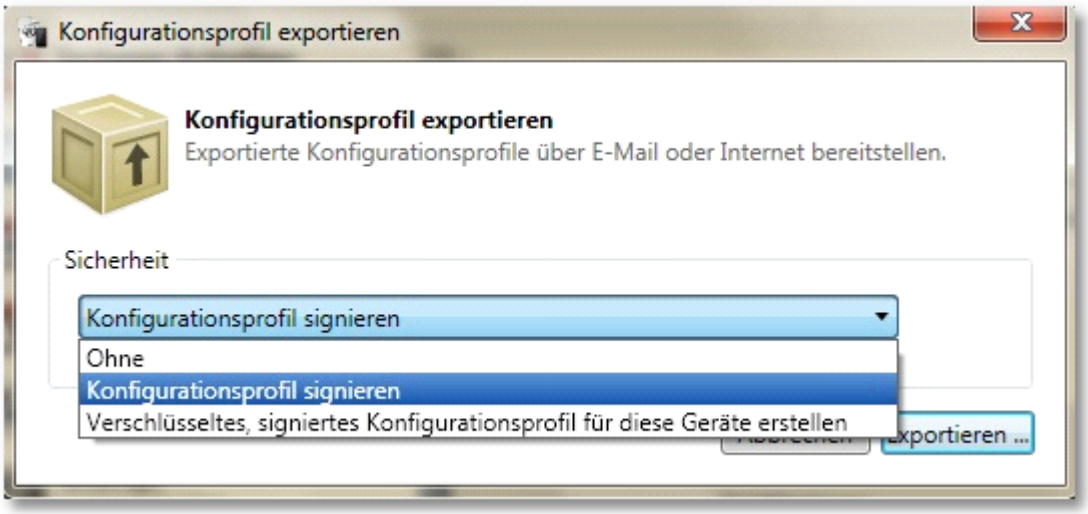
Install configuration profile on the iPad

You can install a configuration profile that you created using the iPhone Configuration Utility on a connected device. You can also send the profile via email to users or deploy it on a website. When users open the email on their device or download the profile of the site, they are automatically prompted to start the installation.

To be able to provide the users the configuration profile, you must export the profile. For this, select the menu item:

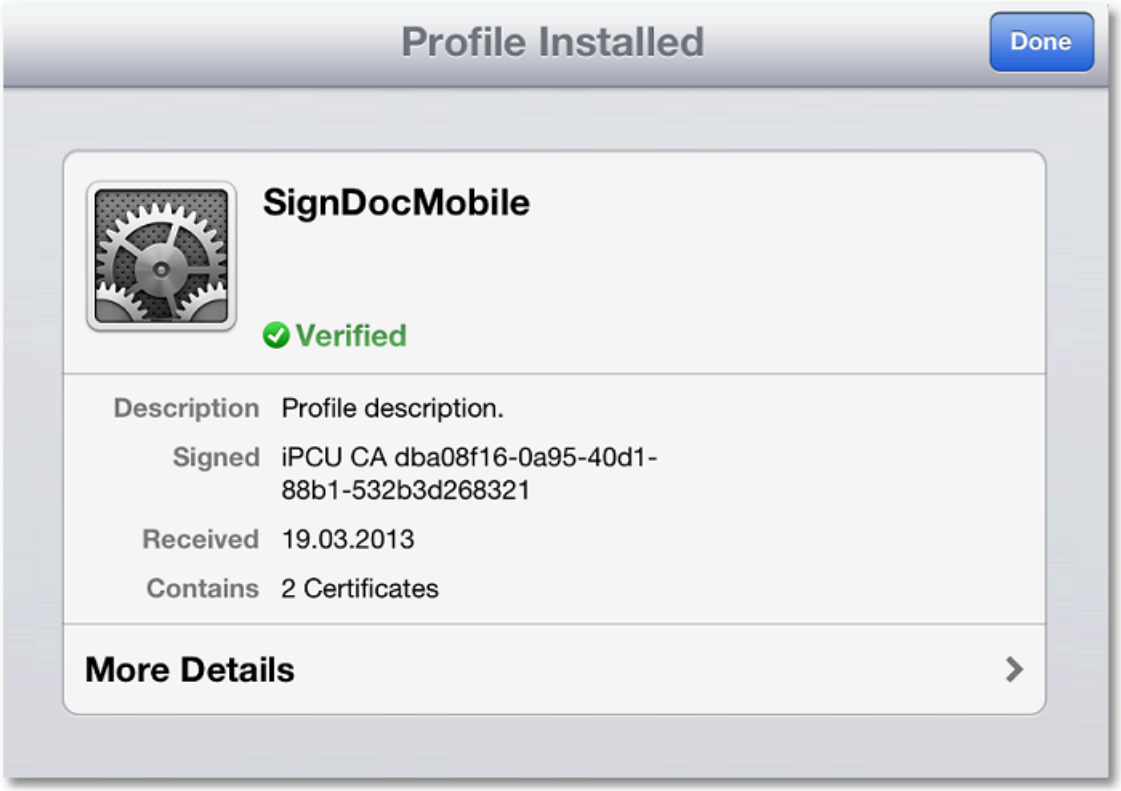
File > Export

You also have the option to "sign" or "encrypt" your profile. Choose from the drop down list of your desired security and click the Export button.



A file with the extension .mobileconfig will be stored on your computer. Make this .mobileconfig file available to your users via email or website.

If you access the configuration profile on your mobile device, the installation will start automatically. You will be informed of the name of the profile and the content and if it is signed.



If you see this dialog, click the Install button to proceed with the installation. When the installation is completed in this dialog the Done button appears. Click the Done button to complete the installation.

On the iOS devices, you can view in the:

Settings > General > Profiles

your installed configuration profiles including certificates.

Logging

Performance log

This section describes the logging.

The entry in XjLog.xml is

```
<logger name="grails.app.bootstrap" level="INFO"/>er.grails.app.bootstrap=INFO
```

Log example

```
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1      n/a 2009-23-11 16:27:57.465
 sdweb.PerformanceLog  - CL SignDoc.onModuleLoad() - initializing client
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:27:57.485 sdweb.PerformanceLog  - CL SignDoc.configurationUpdated() - updated
 client configuration loaded
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:27:59.645 sdweb.PerformanceLog  - CL SignDoc.stateChanged() - document data loaded
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:27:59.967 sdweb.PerformanceLog  - 10-00 LOADER-PHASE(102ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:27:59.968 sdweb.PerformanceLog  - 20-00 PREPARE-PHASE(3344ms) 21-00 PP-CORE-
PREFILL(21ms) 22-00 PP-PLUGIN-COMPLETE(3199ms) 22-02 PP-PLUGIN-STD(1142ms) 22-03 PP-
PLUGIN-POPULATE(2057ms) 23-00 PP-INIT-DOCUMENT(10ms) 24-00 PP-OPTIMIZE-SIZE(106ms)
 26-00 PP-CREATE-KEY(8ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:27:59.968 sdweb.PerformanceLog  - 41-00 IP-SERVERDONE(3453ms) 42-00 IP-FIRSTPAGE-
SENT(6171ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:27:59.968 sdweb.PerformanceLog  - 90-00 MISC-LAST-RENDERED-PAGE(303ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:00.060 sdweb.PerformanceLog  - CL SD_ImagePanel.onLoad() - image of document
 page 1 loaded
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:11.867 sdweb.PerformanceLog  - CL SD_ImagePanel.onLoad() - image of document
 page 4 loaded
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:31.134 sdweb.PerformanceLog  - START SigndocController - printdocument
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:42.560 sdweb.PerformanceLog  - CL FinalizeDocument.actionPerformed() - toolbar
 button 'Send to Archive' clicked
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:45.820 sdweb.PerformanceLog  - CL FinalizeDocument.Dialog.componentSelected() -
 'Finalize Confirm Dialog': button 'Yes' clicked
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:50.056 sdweb.PerformanceLog  - START SigndocController - printdocument
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:55.388 sdweb.PerformanceLog  - 10-00 LOADER-PHASE(102ms)
```

```
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:55.389 sdweb.PerformanceLog  - 20-00 PREPARE-PHASE(3344ms) 21-00 PP-CORE-
PREFILL(21ms) 22-00 PP-PLUGIN-COMPLETE(3199ms) 22-02 PP-PLUGIN-STD(1142ms) 22-03 PP-
PLUGIN-POPULATE(2057ms) 23-00 PP-INIT-DOCUMENT(10ms) 24-00 PP-OPTIMIZE-SIZE(106ms)
 26-00 PP-CREATE-KEY(8ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:55.389 sdweb.PerformanceLog  - 40-00 INTERACTIVE-PHASE(58141ms) 41-00 IP-
SERVERDONE(3453ms) 42-00 IP-FIRSTPAGE-SENT(6171ms) 43-00 IP-ADD-SIG(2163ms) 45-00 IP-
FIELD-UPDATE(165ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:55.389 sdweb.PerformanceLog  - 50-00 FINALIZE-PHASE(3ms) 51-00 FP-
VALIDATE(2017ms) 52-00 FP-STORE-DMS(4016ms)
INFO  199DEB4DAB384849AFF3C704AFB00F4C.worker1  87873-1258990073564 2009-23-11
 16:28:55.389 sdweb.PerformanceLog  - 90-00 MISC-LAST-RENDERED-PAGE(1052ms) 90-03
 PRINT_VIEW(3312ms))
```

Log output description

Log statement	Description
INFO	Log level
199DEB4DAB384849AFF3C704AFB00F4C.worker1	Session ID
87873-1258990073564	Document ID
2009-23-11 16:28:00.060	Date time
sdweb.PerformanceLog	Log class
-	Delimiter
CL/10,20,….	Log phase
LOADER-PHASE(102ms)	Log message

Log phase description

Legend:

Log phase=phase name	Description
CL	Action belongs to Client.
'10-00'='LOADER-PHASE'	Loader phase
'11-00'='LP-CL'	Custom loader
'12-00'='LP-DL'	Document loader ID
'20-00'='PREPARE-PHASE'	Prepare phase
'21-00'='PP-CORE-PREFILL'	Prepare document with core commands.
'22-00'='PP-PLUGIN-COMPLETE'	The complete prepare plugin.
'22-01'='PP-PLUGIN-EXT'	The extended prepare plugin (getdata).
'22-02'='PP-PLUGIN-STD'	The standard prepare plugin (getdata).
'22-03'='PP-PLUGIN-POPULATE'	Populate the document with plugin data.
'23-00'='PP-INIT-DOCUMENT'	Initialize the document.
'24-00'='PP-OPTIMIZE-SIZE'	Optimize the document size.

Log phase=phase name	Description
'25-00'='PP-READONLY-MODE'	Read Only Mode
'26-00'='PP-CREATE-KEY'	Create the signing key.
'30-00'='CONVERSION-PHASE'	Convert the document.
'40-00'='INTERACTIVE-PHASE'	Interactive phase (GUI)
'41-00'='IP-PREP_SIG_PLUGIN'	Prepare browser plugin for signature.
'42-00'='IP-PREP_IMG_PLUGIN'	Prepare browser plugin for image capturing.
'43-00'='IP-ADD-SIG'	A signature was added.
'44-00'='IP-DEL-SIG'	A signature was deleted.
'45-00'='IP-FIELD-UPDATE'	An interactive field was updated.
'45-01'='IP-FIELD-VALIDATE-UPDATE'	Validation before field update
'46-00'='‘IP_VALIDATE‘	Document validation before archiving.
'47-00'='‘IP_ADD_IMAGE‘	Adding an image to the document.
'50-00'='FINALIZE-PHASE'	Finalize phase
'52-00'='FP-STORE-DMS'	Store the PDF in the DMS.
'90-00'='MISC-LAST-RENDERED-PAGE'	Rendering a single page.
'90-01'='MISC1'	undefined
'90-02'='MISC-DOWNLOAD-DOCUMENT'	Downloading the document.
'90-03'='PRINT_VIEW'	The time the printview is visible.
'90-04'='MISC_SERVER_DONE'	Prepare on server is done. Delivered GUI page.
'90-05'='MISC_FIRSTPAGE_DONE'	GUI can be used. 1st page is streamed to client.
'99-03'='MISC-3'	undefined
'99-03'='MISC-4'	undefined
'99-03'='MISC-5'	undefined
'99-03'='MISC-6'	undefined
'99-03'='MISC-7'	undefined
'99-03'='MISC-8'	undefined
'99-03'='MISC-9'	undefined

Managing key pairs for encryption of biometric data

Background

SignDoc Web encrypts the biometric data of a signature using the public key of a key pair.

The private key must be kept secret and is used for decrypting the biometric data and checking whether the document was signed with that biometric data.

A default public key is stored in a file (0001-public.key) which is shipped with SignDoc Web in the war file. After deploying the war file the public key file can be found in the directory:

- WEB-INF/lib/native/win32 (Windows)
- or WEB-INF/lib/native/linux-i386 (Linux)

The private key is encrypted with a passphrase (ASCII only) and stored in a file which is deployed to a special workplace where signatures are to be verified.

Install the new key pair

The name of the public key file (default:0001-public.key) that is used to encrypt the biometric data can be set via sdweb_config.groovy setting sdweb.defaults.defaultpublickey.name, e.g.

```
sdweb.defaults.defaultpublickey.name = 'my-public.key'
```

The public key file is expected from SignDoc Web in the %SDWEB_HOME%/resources directory (%SDWEB_HOME% is set via environment variable SDWEB_HOME).

Note The location of the resource directory can be changed via sdweb.resources.dir setting in sdweb_config.groovy.

It is essential to encrypt biometric data asymmetrically and to keep the private key secret. This should be done via RSA cryptosystem. To create the RSA key pair, you can use either KeyTool (which will use a proprietary file format for encrypted private keys) or any tool that creates an RSA key pair and uses PKCS #1 format (DER or PEM) for the public key and PKCS #12 format for the private key. Alternatively, the public key can also be specified as X.509 certificate (DER or PEM).

Create a key with KeyTool

KeyTool is available for the operating systems Windows and Linux.

For Windows use the file KeyTool.exe and for Linux use the file KeyTool.

Create a key pair

To create a private/public key pair, run KeyTool this way:

```
KeyTool create [-p PASSPHRASE] [BASE]
```

PASSPHRASE is the passphrase used to protect the private key. The private key will only be usable if the correct passphrase is entered at the time biometric data is decrypted. If the -p option is not used, KeyTool will ask for a passphrase. If the passphrase is empty, the private key won't be encrypted (that's not recommended). Note that the passphrase can be changed later on as described below.

If BASE is specified, files BASE-private.key and BASE-public.key will be written. If BASE is not specified, SignDoc4ADS will be used, that is, files SignDoc4ADS-private.key and SignDoc4ADS-public.key will be written.

Example

```
KeyTool create -p "Vveri Zekkret" new
```

Change the passphrase of a private key

To change the passphrase of a private key, run KeyTool this way:

```
KeyTool crypt [-pi PASSPHRASE] [-po PASSPHRASE] INPUT OUTPUT
```

INPUT is the pathname of an existing private key file; that file will be read. OUTPUT is the pathname of the new private key file to be written. INPUT and OUTPUT must not reference the same file.

There are two passphrases, the old one (of the input file) and the new one (of the output file). The old one is specified with -pi, the new one with -po. sdsdakey will ask you for any passphrases not specified on the command line.

Example

```
KeyTool crypt -pi "Vveri Zekkret" -po "eeven moRR seecred" new-private.key
newer-private.key
```

Create a key pair using OpenSSL

Using OpenSSL on the command line you'd first need to generate a public and private key (you could password protect this file using the -passout argument). Read OpenSSL documentation for more information (https://www.openssl.org/).

```
openssl genrsa -out private.pem -aes256 2048
```

This creates a key file called private.pem that uses 2048 bits. This file actually has both the private and public keys, so you should extract the public one from this file:

```
openssl rsa -in private.pem -out public.der -outform DER -pubout
```

Finally you have to create a PKCS 12 archive file:

```
openssl pkcs12 -export -inkey private.pem -nocerts -out private.der
```

The export password must be non-empty. The public key will be written to public.der, the private key will be written to private.der and the file private.pem can be deleted.

Multi-instance configuration

JAVA JAR files using native code via JNI

In general, some JAR files of the sdweb.war file need to be moved to a common lib directory used by all web contexts of a J2EE application server.

> **Important** This is a must, if multiple contexts of a J2EE application server use the same JAR file accessing native JNI code.

This is optional, if sdweb is the only context in a J2EE application server.

The files to be moved to the common directory are basically all JAR files using JAVA-JNI to load native code. This is a constraint from the Sun JAVA JRE.

Example for the common/lib directory:

```
Tomcat: <TOMCAT_DIR>/common/lib
```

SDWEB core JAR files:

```
* WEB-INF\lib\splm2jni.jar [Kofax License Management]
* WEB-INF\lib\sppdfjni_*.jar (e.g. sppdfjni_4.5_9.jar) [Kofax PDF libs]
* WEB-INF\lib\SPSignDoc_*.jar (e.g. SPSignDoc_4.5_9.jar) [Kofax SignDoc libs]
* WEB-INF\lib\sputiljni.jar [Kofax utilities]
```

Optional JAR files (maybe not installed):

```
* WEB-INF\lib\SPSignware_*.jar (e.g. SPSignware_19.jar) [Kofax SignWare]
```

Instance-based configuration

When SignDoc Web is running in multiple instances in an Application container, it's possible to configure the instances independently.

Procedure

Each SDWEB instance process has to set the JAVA property:

```
-DSDWEB_INSTANCEID=<instance name>
```

Example

```
-DSDWEB_INSTANCEID=inst1
```

Now it is possible to have for the following configuration files an independent configuration file in a separate location:

- sdweb_config.groovy
- XjLog.xml

The instance configuration files are located in the directory and overwrite the default files/setting when present:

```
%SDWEB_HOME%/conf/<instance name>
```

Example

```
C:\Users\auser\sdweb_home\conf\inst1\XjLog.xml
```

Page pre-fetching in WebView

For a better user acceptance the loading of documents can be enhanced by using the pre-fetching mechanism in SignDoc Web.

If this feature is enabled all remaining pages of a document are loaded at once after the document is opened in the WebView.

There are mainly two different configurable options when using pre-fetching:

1. Blocking:
   The document is loaded completely (or the maximum numbers of pages are pre-fetched) before user can start editing the document
2. Not blocking:
   The user can start editing the document during pre-fetching is done in the background

In both configuration cases users are informed about remaining number of pages by a loading bar.

Depending on the configured option the loading bar appears at the bottom of the WebView screen (not blocking) or in the center of the WebView (blocking).

Configuration entries for pre-fetching:

1. signdoc_configuration.xml:

```
<component id="Miscellaneous">
   ...
   <element id="Page.Prefetch">
   <!--
   Specify the page pre-fetching mode:
   0 - page pre-fetching is disabled
   1 - page pre-fetching is enabled.
       User actions are blocked until all pages are loaded.
   2 - page pre-fetching is enabled.
       User actions are immediately available and pages are loaded in the background.
   -->
   <parameter key="Mode" value="0"/>
   <!--
   Specify the page pre-fetching strategy
   0 - all document pages are pre-fetched starting with page 1 (if number of document
       exceeds the configured maximum of allowed pre-fetched pages all document pages
 up to this value are pre-fetched)
    ... (more strategies will be available in the future)
   -->
   <parameter key="Strategy" value="0"/>
   <parameter key="Label" propertyFile="language" propertyKey="Page.Prefetch.Label"/>
   <!-- The maximal number of pages which can be pre-fetched -->
   <parameter key="Max" value="500"/>
   </element>
```

2. mobile_configuration.xml:

```
<component id="Miscellaneous">
...
     <element id="Page.Prefetch">
     <!--
     Specify the page pre-fetching mode:
     0 - Page pre-fetching is disabled
     1 - Page pre-fetching is enabled when document is loaded.
     User actions are blocked until all pages are loaded.
     2 - Page pre-fetching is enabled when document is loaded.
     User actions are immediately available and pages are loaded in the background.
     3 - Page pre-fetching is enabled when document is loaded or screen is rotated in
 PINCH_SPREAD/FIT_TO_WIDTH zoom mode.
     User actions are blocked until all pages are loaded.
     4 - Page pre-fetching is enabled when document is loaded or screen is rotated in
 PINCH_SPREAD/FIT_TO_WIDTH zoom mode.
     User actions are immediately available and pages are loaded in the background.
```

```
    -->
    <parameter key="Mode" value="0"/>
    <!--
    Specify the page pre-fetching strategy
    0 - all document pages are pre-fetched starting with page 1 (if number of document
    exceeds the configured maximum of allowed pre-fetched pages all document pages up
to this value are pre-fetched)
    ... (more strategies will be available in the future)
     -->
    <parameter key="Strategy" value="0"/>
    <parameter key="Label" propertyFile="language" propertyKey="Page.Prefetch.Label"/>
    <!-- The maximal number of pages which can be pre-fetched -->
    <parameter key="Max" value="500"/>
    </element>
```

Reduce network data

If clients use high resolution screens, the data that is sent over the network to display the rendered document pages is quite big. This hurts especially in mobile scenarios. Depending on the contents of the document and the resolution of the screen, an image of a single page can be, in bad cases, multiple hundred KBytes.

There are different strategies to reduce the amount of data sent to the client.

Default page image formats

Desktop Browser (Internet Explorer, Chrome, Firefox, Edge)

- Format: PNG (with zip compression)
- Colors: 24 Bit/Pixel (true color)

Mobile Devices (SignDoc Mobile, Safari Mobile, Chrome Mobile)

- Format: GIF
- Colors: Dithering, 8 Bit/Pixel

Strategy 1 - Use jpeg and/or gif format

The jpeg format reduces the size of many images significantly (size reduction depends on jpeg quality setting). The downside is that it produces visible artifacts for high-contrast areas. Example: small black text on white background. The quality of the jpeg images can be adjusted. Smaller quality numbers mean a smaller image size but also more artifacts.

The gif format usually reduces the file size significantly (~30% size reduction). The downside is that color loss occurs and photos might look not so nice.

Usage in sdweb_config.groovy

```
sdweb.gui.render.format="jpeg" // default render format
sdweb.gui.render.jpeg.quality=60 // adjust jpeg quality [range: 0-100]
sdweb.gui.render.format_mobile="gif" // format used for known mobile browsers
```

Strategy 2 - Apply predefined indexed color modes

If Strategy 1 is not sufficient, custom predefined indexed color modes can be used. This means that the color palette is limited to a fixed color number.

This makes only sende with file formats that support a color palette like PNG and GIF.

The Server provides predefined indexed color modes that can be used by configuration.

Predefined indexed color modes (grey)

- CM_2_GREY_LEVELS, CM_BW
  Minimum image size! (~90% size reduction*)
- Black and White Images. All Colors are removed.
  CM_3_GREY_LEVELS
- Renders the page image in 3 evenly distributed grey levels
  CM_4_GREY_LEVELS
- Renders the page image in 4 evenly distributed grey levels
  CM_5_GREY_LEVELS
- Renders the page image in 5 evenly distributed grey levels
  CM_6_GREY_LEVELS
- Renders the page image in 6 evenly distributed grey levels
  CM_7_GREY_LEVELS
- Renders the page image in 7 evenly distributed grey levels
  CM_8_GREY_LEVELS
  No information loss except color! (~80% size reduction*)
  Renders the page image in 8 evenly distributed grey levels

Predefined indexed color modes (color)

- CM_6_COLORS (~83% size reduction*)
  Produces often good or at least acceptable color images with a minimum image size.

*compared with a PNG 24 Bit image. Data is from a real customer installation.

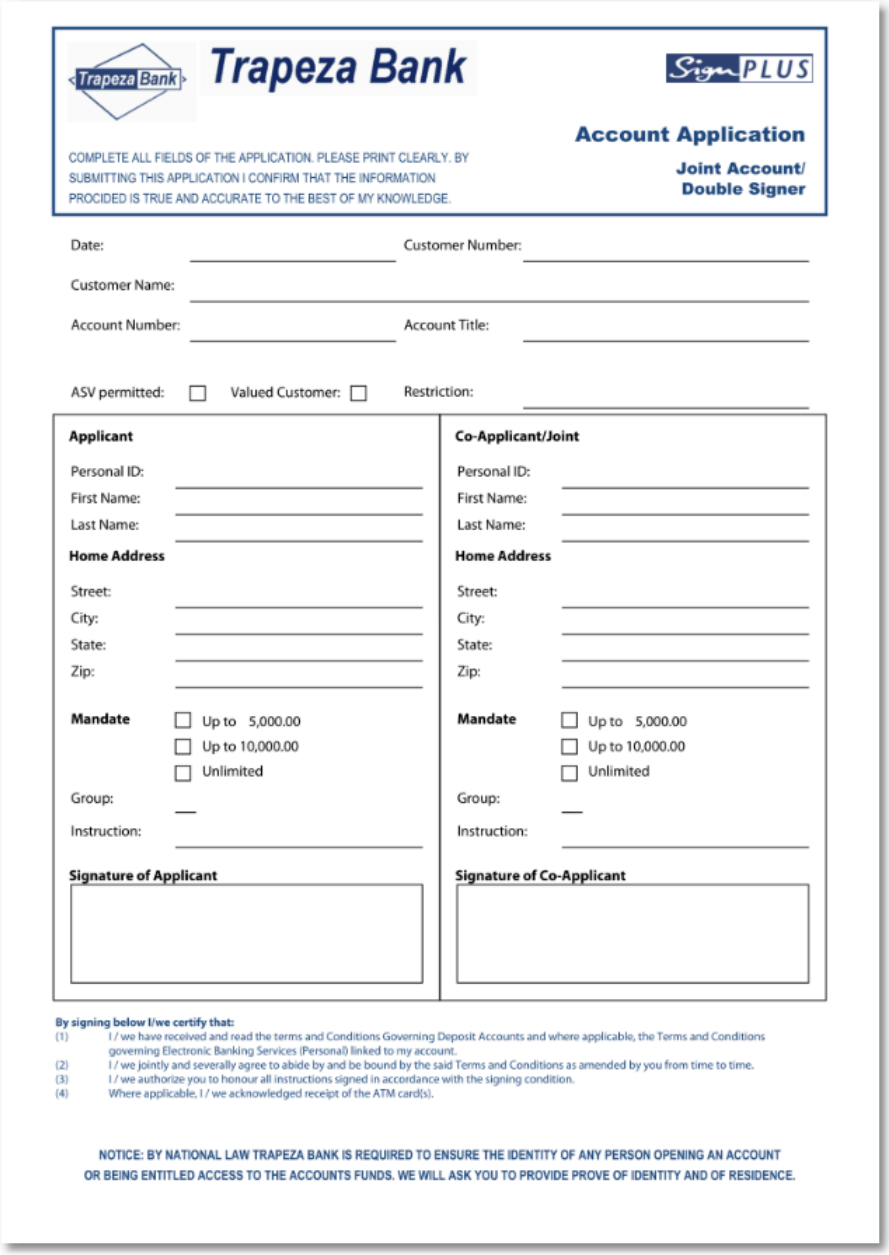
Usage in sdweb_config.groovy

```
sdweb.gui.render.index_color_model="CM_8_GREY_LEVELS"
sdweb.gui.render.index_color_model_desktop="CM_8_GREY_LEVELS"
sdweb.gui.render.index_color_model_mobile="CM_2_GREY_LEVELS"
```

Important Don’t use an indexed color mode preset with a setting sdweb.gui.render.format="jpeg" or sdweb.gui.render.format_mobile="jpeg"! Doing so it will rerender the already reduced and dithered image again as jpeg. This will lead to a much bigger image and many ugly artifacts!

Example

Default



Trapeza Bank

SignPLUS

Account Application

Joint Account/ Double Signer

COMPLETE ALL FIELDS OF THE APPLICATION. PLEASE PRINT CLEARLY. BY SUBMITTING THIS APPLICATION I CONFIRM THAT THE INFORMATION PROCIDED IS TRUE AND ACCURATE TO THE BEST OF MY KNOWLEDGE.

Date: _____ Customer Number: _____

Customer Name: _____

Account Number: _____ Account Title: _____

ASV permitted: ☐ Valued Customer: ☐ Restriction: _____

Applicant	**Co-Applicant/Joint**
Personal ID:	Personal ID:
First Name:	First Name:
Last Name:	Last Name:
Home Address	**Home Address**
Street:	Street:
City:	City:
State:	State:
Zip:	Zip:
Mandate ☐ Up to 5,000.00 ☐ Up to 10,000.00 ☐ Unlimited	**Mandate** ☐ Up to 5,000.00 ☐ Up to 10,000.00 ☐ Unlimited
Group:	Group:
Instruction:	Instruction:
Signature of Applicant	**Signature of Co-Applicant**

By signing below I/we certify that:

(1) I / we have received and read the terms and Conditions Governing Deposit Accounts and where applicable, the Terms and Conditions governing Electronic Banking Services (Personal) linked to my account.

(2) I / we jointly and severally agree to abide by and be bound by the said Terms and Conditions as amended by you from time to time.

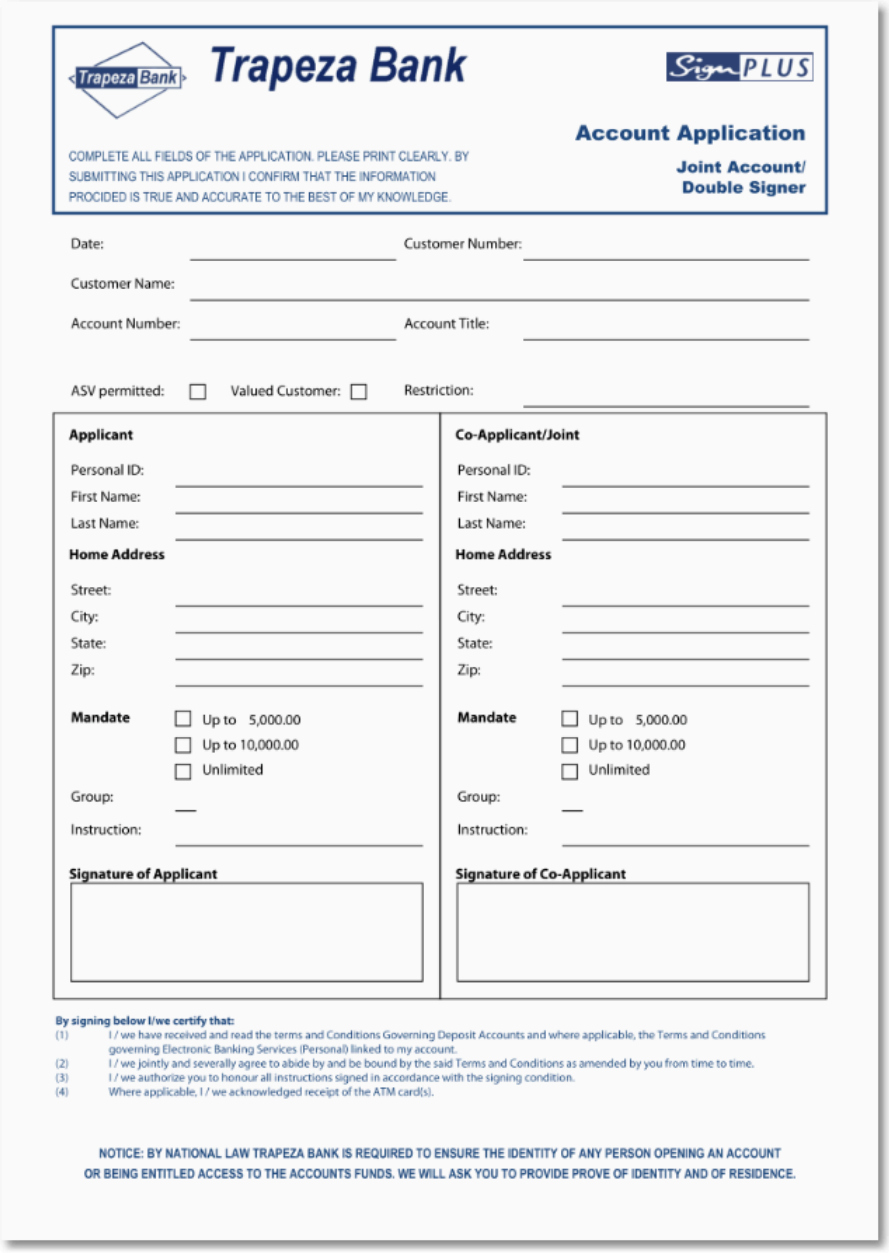
(3) I / we authorize you to honour all instructions signed in accordance with the signing condition.

(4) Where applicable, I / we acknowledged receipt of the ATM card(s).

NOTICE: BY NATIONAL LAW TRAPEZA BANK IS REQUIRED TO ENSURE THE IDENTITY OF ANY PERSON OPENING AN ACCOUNT OR BEING ENTITLED ACCESS TO THE ACCOUNTS FUNDS. WE WILL ASK YOU TO PROVIDE PROVE OF IDENTITY AND OF RESIDENCE.

Example

Default GIF



Trapeza Bank

SignPLUS

Account Application

Joint Account/ Double Signer

COMPLETE ALL FIELDS OF THE APPLICATION. PLEASE PRINT CLEARLY. BY SUBMITTING THIS APPLICATION I CONFIRM THAT THE INFORMATION PROCIDED IS TRUE AND ACCURATE TO THE BEST OF MY KNOWLEDGE.

Date: _____ Customer Number: _____

Customer Name: _____

Account Number: _____ Account Title: _____

ASV permitted: ☐ Valued Customer: ☐ Restriction: _____

Applicant	**Co-Applicant/Joint**
Personal ID:	Personal ID:
First Name:	First Name:
Last Name:	Last Name:
Home Address	**Home Address**
Street:	Street:
City:	City:
State:	State:
Zip:	Zip:
Mandate ☐ Up to 5,000.00 ☐ Up to 10,000.00 ☐ Unlimited	**Mandate** ☐ Up to 5,000.00 ☐ Up to 10,000.00 ☐ Unlimited
Group:	Group:
Instruction:	Instruction:
Signature of Applicant	**Signature of Co-Applicant**

By signing below I/we certify that:

(1) I / we have received and read the terms and Conditions Governing Deposit Accounts and where applicable, the Terms and Conditions governing Electronic Banking Services (Personal) linked to my account.

(2) I / we jointly and severally agree to abide by and be bound by the said Terms and Conditions as amended by you from time to time.

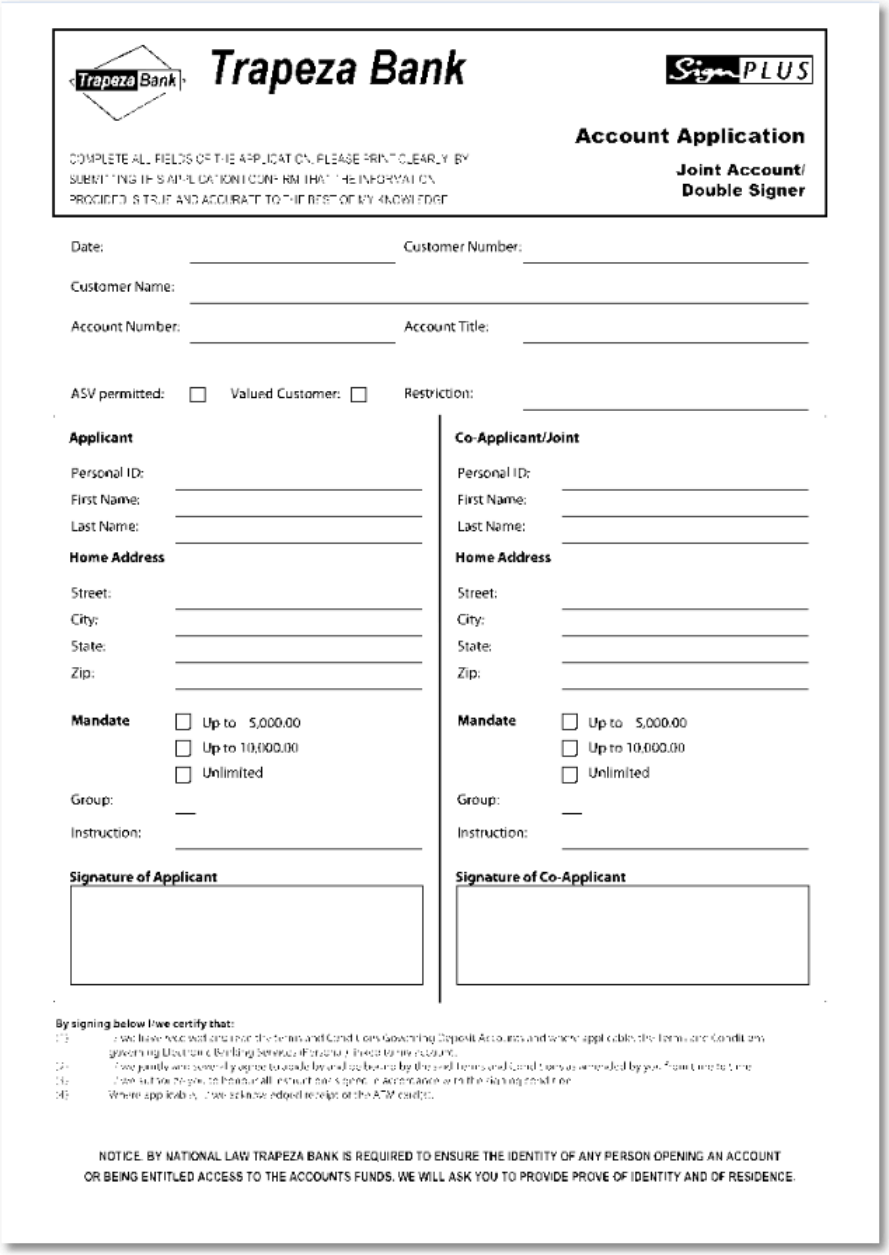
(3) I / we authorize you to honour all instructions signed in accordance with the signing condition.

(4) Where applicable, I / we acknowledged receipt of the ATM card(s).

NOTICE: BY NATIONAL LAW TRAPEZA BANK IS REQUIRED TO ENSURE THE IDENTITY OF ANY PERSON OPENING AN ACCOUNT OR BEING ENTITLED ACCESS TO THE ACCOUNTS FUNDS. WE WILL ASK YOU TO PROVIDE PROVE OF IDENTITY AND OF RESIDENCE.

Example

CM_2_COLORS



Trapeza Bank

SignPLUS

Account Application

Joint Account/ Double Signer

COMPLETE ALL FIELDS OF THE APPLICATION. PLEASE PRINT CLEARLY. BY SUBMITTING THIS APPLICATION I CONFIRM THAT THE INFORMATION PROVIDED IS TRUE AND ACCURATE TO THE BEST OF MY KNOWLEDGE

Date: _____ Customer Number: _____

Customer Name: _____

Account Number: _____ Account Title: _____

ASV permitted: ☐ Valued Customer: ☐ Restriction: _____

Applicant

Personal ID: _____

First Name: _____

Last Name: _____

Home Address

Street: _____

City: _____

State: _____

Zip: _____

Mandate

☐ Up to 5,000.00

☐ Up to 10,000.00

☐ Unlimited

Group: _____

Instruction: _____

Signature of Applicant

Co-Applicant/Joint

Personal ID: _____

First Name: _____

Last Name: _____

Home Address

Street: _____

City: _____

State: _____

Zip: _____

Mandate

☐ Up to 5,000.00

☐ Up to 10,000.00

☐ Unlimited

Group: _____

Instruction: _____

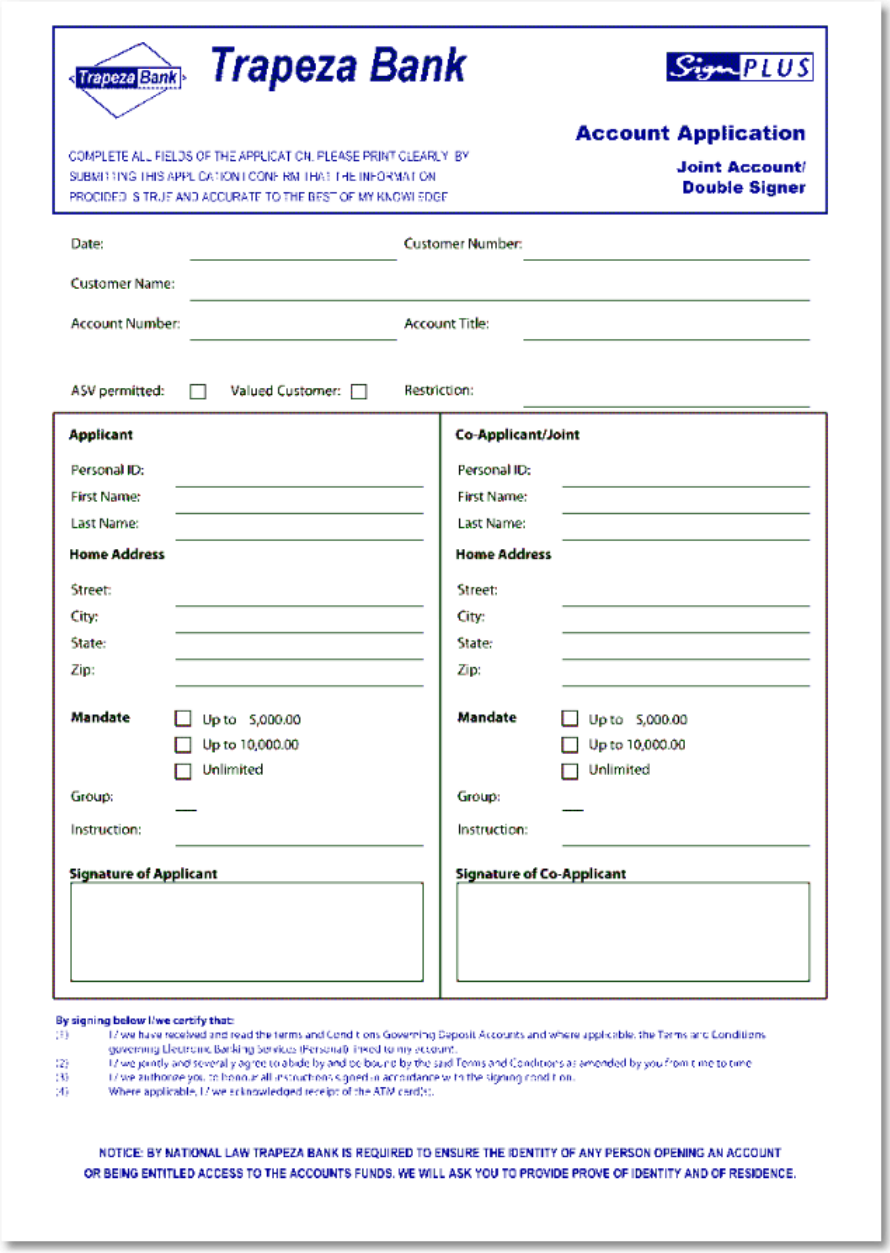
Signature of Co-Applicant

By signing below I/we certify that:

NOTICE. BY NATIONAL LAW TRAPEZA BANK IS REQUIRED TO ENSURE THE IDENTITY OF ANY PERSON OPENING AN ACCOUNT OR BEING ENTITLED ACCESS TO THE ACCOUNTS FUNDS. WE WILL ASK YOU TO PROVIDE PROVE OF IDENTITY AND OF RESIDENCE.

Example

CM_6_COLORS



Trapeza Bank

SignPLUS

Account Application

Joint Account/ Double Signer

COMPLETE ALL FIELDS OF THE APPLICATION. PLEASE PRINT CLEARLY. BY SUBMITTING THIS APPLICATION I CONFIRM THAT THE INFORMATION PROVIDED IS TRUE AND ACCURATE TO THE BEST OF MY KNOWLEDGE

Date: _____ Customer Number: _____

Customer Name: _____

Account Number: _____ Account Title: _____

ASV permitted: ☐ Valued Customer: ☐ Restriction: _____

Applicant	**Co-Applicant/Joint**
Personal ID:	Personal ID:
First Name:	First Name:
Last Name:	Last Name:
Home Address	**Home Address**
Street:	Street:
City:	City:
State:	State:
Zip:	Zip:
Mandate ☐ Up to 5,000.00 ☐ Up to 10,000.00 ☐ Unlimited	**Mandate** ☐ Up to 5,000.00 ☐ Up to 10,000.00 ☐ Unlimited
Group:	Group:
Instruction:	Instruction:
Signature of Applicant	**Signature of Co-Applicant**

By signing below I/we certify that:

(1) I/we have received and read the terms and conditions Governing Deposit Accounts and where applicable, the Terms and Conditions governing Electronic Banking Services (Personal) linked to my account.

(2) I/we jointly and severally agree to abide by and be bound by the said Terms and Conditions as amended by you from time to time

(3) I/we authorise you to honour all instructions signed in accordance with the signing condition.

(4) Where applicable, I/we acknowledged receipt of the ATM card(s).

NOTICE: BY NATIONAL LAW TRAPEZA BANK IS REQUIRED TO ENSURE THE IDENTITY OF ANY PERSON OPENING AN ACCOUNT OR BEING ENTITLED ACCESS TO THE ACCOUNTS FUNDS. WE WILL ASK YOU TO PROVIDE PROVE OF IDENTITY AND OF RESIDENCE.

Sign2Phone administration

Configuration

If SignDoc should use mobile devices as the only device for signature capturing you can define this in the SignDoc configuration file. To do this set:

```
sdweb.browserplugin.padclass="SPTabletRemoteTablet"
```

Connectivity

A prerequisite to run Sign2Phone with Kofax SignDoc Device Support is, that the mobile device and the PC access the same network.

We recommend to use existing WiFi infrastructure to connect your mobile device with your Kofax SignDoc Device Support installation. Make sure your PC and mobile device reside on the same network.

Nevertheless it is possible to set up local network environment between your iOS device and your computer on which the Kofax SignDoc Device Support is installed.

There are 3 possibilities to set up such local network environment (depending on your iOS device and your mobile contract).

Establish connectivity between PC and mobile device - via Wi-Fi or USB Cable or Bluetooth:

- WiFi: Enable personal hotspot functionality on your iOS device and connect your computer to this WiFi
- USB Cable: Enable personal hotspot functionality on your iOS device and use the USB cable to connect to your computer
- Bluetooth: Enable personal hotspot functionality on your iOS device and connect it to your computer via Bluetooth

The mobile device communicates with the application via TCP Port 46998 (service name is sp-remotetablet) and UDP Port 5353 (for Bonjour service). A secure connection can be established by limiting the mobile device connectivity to only that port. All other ports may be blocked for security. This will enable a secure solution via WiFi between the mobile devices and the desktop application.

Signer-specific certificates

Description

This feature allows to define a specific certificate that should be used, when applying the digital signature to a certain signature field.

Prerequisites

- The certificates must be in PKCS#12 format
- The certificates must all have the same password
- The PKCS#12 file must contain only 1 certificate
- The file name syntax is:
  certificate_id.p12
  Example
  my_certificate.p12
- The files must be put in the directory:
  SDWEB_HOME/conf/user_certificates
  Example
  /var/sdwebhome/conf/user_certificates/my_certificate.p12

Useful tools

Portecle: Create PKCS#12 certificate stores

Usage

Settings in sdweb_config.goovy configuration file:

```
sdweb.signerspecific.store.pkcs12.password="common password of the signer
specific certificates"
```

Prepare the signature field:

To enable a signature field to use a specific certificate, the document has to be prepared using a command statement in the form request.

The in-lined key/value pair cert=certificate_id defines the certificate to use, when signing the signature field.

Syntax

```
<input name="cmd_1" value="name=SignatureFixed1|cert=certificate_id|..."/>
```

Example

```
<form action="http://localhost:8080/sdweb/load/byurl" method="post" >
  <input name="docurl" value="url_of_the_document" type="hidden"/>
  <input name="cmd_1" value="name=SignatureFixed1|cert=my_certificate|page=1|bottom=10|
left=10|width=100|height=50|label=SignatureFixed1|type=formfield|subtype=signature"
 type="hidden"/>
  <input type="submit" value="OPEN DOCUMENT"/>
</form>
```

In this example, a signature field will be inserted at the specified coordinates. When the signature field is signed in SignDoc Web, the server will use the file:

```
SDWEB_HOME/conf/user_certificates/my_certificate.p12
```

when applying the digital signature. If the file is missing, the configured server certificate or a one time certificate will be used.

Kofax print plugin integration

Description

Although SignDoc Web is used to minimize and/or eliminate paper work, some customers require a good print functionality. With this version, SignDoc Web provides 3 print methods the customer can choose form.

- **Printing method 1:** Browser printing
  Pros and cons
  + No dependency on the client environment
  + Works with every browser in every operating system
  + The document stays always on the server side
  - Print quality is limited
  - Needs a couple of seconds until printing is ready
  - Might require some print margin adjustments in the browser
- **Printing method 2:** Use registered mimetype handler application
  Pros and cons
  + Possibly perfect printout quality (depends on the MIME type handler application)
  - Dependency on the MIME type handler application e.g. Adobe Reader
  - The document is possibly loaded on the client
  - Possibly a local temp file is created
  - Possibly a local copy of the file can be created by the user
- **Printing method 3:** Use the Kofax Print Plugin
  Pros and cons
  + Integrated with SignDoc Web
  + Perfect printout
  + Fast display
  + Simple and easy user interface
  + Limits actions of the user to an absolute minimum
  - Requires Kofax SignDoc Device Support and browser extension installation on client side
  - A local tempfile is created while displaying document
  - Works only under Windows (currently only with 32 Bit Browsers)

Requirements

Note Possibly, there will be in future a setup to handle this part of the client installation.

- Kofax SignDoc Device Support
- Browser extension

Configuration (server side)

The print method is configured in sdweb_config.groovy by the following properties:

- **sdweb.print.method.impl** (string): The print method to use. Default value: "printdocument"
  Example
  Method 1 (default): sdweb.print.method.impl="printdocument
  Method 2: sdweb.print.method.impl="print"
  Method 3: sdweb.print.method.impl="printplugin"
- **sdweb.printplugin.timeout** Timeout in ms for opening the document to print before an error is reported. Default value: 25000
- **sdweb.printplugin.width** Initial width of the print plugin. Default value: 700
- **sdweb.printplugin.height** Initial height of the print plugin. Default value: 500
- **sdweb.printplugin.optionsfile** The options file that defines the layout of the print plugin's GUI. Details to the syntax are available in the documentation of the SBPlugin component. The default layout can be found and edited in SDWEB_HOME/resources/PrintOptions.xml. Default value: "PrintOptions.xml"

TSA functionality

According to the RFC 3161 standard, a trusted timestamp is a timestamp issued by a trusted third party (TTP) acting as a Time Stamping Authority (TSA).

It is used to prove the existence of certain data before a certain point (e.g. contracts, research data, medical records,...) without the possibility that the owner can backdate the timestamps.

There are basically 2 cases.

Case 1: If a document is already prepared according to the PDF standard to use a specific TSA it will be used automatically when signing. There are no further configuration setting required.

Case 2: A simple unprepared document should use a TSA when signing.

The following configuration settings can be used:

Option	Description
Option 1	TSA setting is provided in sdweb_config.groovy (see details below) `sdweb.document.signature.tsa.config="TSA-CONFIGURATION-STRING"`
Option 2	Document specific TSA setting (optional) Further, it is possible, to set the TSA configuration via the metadata key: SDWEB_METADATA_DOCUMENT_TSA_CONFIG This will override the global setting (option 1). To enable this, the following setting has to be made in sdweb_config.groovy: `sdweb.document.signature.tsa.use_metadata_config=true`

Options for the TSA-CONFIGURATION-STRING have to be separated by the pipe symbol (|).

Setting it via the command option as metdata for the document will require a change of the command separator via the sdweb_config.groovy "sdweb.command.list_separator" parameter.

Parameter options available are:

- **TimeStampServerUR** (required): The URL of an RFC 3161 time-stamp server. If string parameter "Timestamp" is empty and string parameter "TimeStampServerURL" is non-empty, a time stamp will be obtained from a time-stamp server. The scheme of the URL must be either "http" or "https". The time-stamp server URL specified by the document's signature field seed value dictionary overrides the "TimeStampServerURL" parameter. An error will be returned by SignDocDocument.addSignature() if a time-stamp server is to be used and integer parameter "Method" is not m_digsig_pkcs7_detached or m_digsig_pkcs7_sha1.

  See also integer parameter

  TimeStampServerTimeout

  and string parameters:

  TimeStampClientCertificatePath

  TimeStampClientKeyPath

  TimeStampServerPassword

  TimeStampServerTrustedCertificatesPath

  TimeStampHashAlgorithm

  TimeStampServerUser
- **TimeStampClientCertificatePath** (string, optional): The pathname of a file containing the certificate in PEM format for authenticating to an RFC 3161 time-stamp server over HTTPS. If the parameter is non-empty, string parameter "TimeStampClientKeyPath" must also be set. If the value is empty, the client won't authenticate itself.

  The default value is empty.

  See also string parameters:

  TimeStampServerURL

  TimeStampClientKeyPath

  TimeStampServerTrustedCertificatesPath
- **TimeStampClientKeyPath** (string, optional): The pathname of a file containing the private key in PEM format for authenticating to an RFC 3161 time-stamp server over HTTPS. If the parameter is non-empty, string parameter "TimeStampClientCertificatePath" must also be set. If the value is empty, the client won't authenticate itself.

  The default value is empty.

  See also string parameters:

  TimeStampServerURL

  TimeStampClientKeyPath

  TimeStampServerTrustedCertificatesPath
- **TimeStampServerPassword** (string, optional): The password for Basic/Digest HTTP authentication to the time-stamp server. Non-ASCII values probably don't work. If this parameter is set, string parameter "TimeStampServerUser" must also be set.

- **TimeStampServerTrustedCertificatesPath** (string, optional): The pathname of a file containing trusted root certificates in PEM format for authenticating RFC 3161 time-stamp servers over HTTPS. If the value is empty, time-stamp servers won't be authenticated.

  The default value is empty.

  See also string parameters:

  TimeStampServerURL

  TimeStampClientCertificatePath

  TimeStampClientKeyPath
- **TimeStampServerUser** (string, optional): The user name for Basic/Digest HTTP authentication to the time-stamp server. Non-ASCII values probably don't work. If this parameter is set, string parameter "TimeStampServerPassword" must also be set.
- **TimeStampServerTimeout** (integer, optional): Time out in milliseconds for retrieving a time stamp from an RFC 3161 time-stamp server. The value must be positive.

  The default value is 10000.

  See also string parameter:

  TimeStampServerURL
- **TimeStampHashAlgorithm** (string, optional): Hash algorithm for RFC 3161 time-stamps. The accepted values are 0 (default currently SHA-256), 1 (SHA-1) or 2 (SHA-256).

  The default value is 0.

  See also string parameter

  TimeStampServerURL

sdweb_config.groovy string examples:

1. Public TSA Server (Example only!, No guarantee that this server is working or is online all the time):

   ```
   sdweb.document.signature.tsa.config="TimeStampServerURL=http://
   zeitstempel.dfn.de"
   ```
2. Public TSA Server using SHA-256 (Example only!, No guarantee that this server is working or is online all the time):

   ```
   sdweb.document.signature.tsa.config="TimeStampServerURL=https://bteszt.e-
   szigno.hu/tsa|TimeStampServerUser=teszt|TimeStampServerPassword=teszt|
   TimeStampHashAlgorithm=2"
   ```

Additional notes

Note that the temporary files under "%TOMCAT_HOME%\temp\" should be monitored regularly.

The administrator may need to delete the files if there is an excessive number of files.

Chapter 5

Standard plugins

File DMS plugin

The File DMS plugin is a SignDoc Web standard plugin that stores the documents on a local file system or mounted network drive.

Plugin setup

The plugin can be loaded and so used by adding the plugins class name to the 'loadlist' property in the Configuration file sdweb_config.groovy.

Loading the plugin

Example

```
sdweb.plugins.loadlist = ['de.softpro.sdweb.plugins.impl.FileDms']
```

Configuration

It can be configured to create different files regarding the archived document.

It is possible to request up to 5 different files by configuration in sdweb_config.groovy with prefix 'sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".' followed by the appropriate setting.

1. The (archived) document file (pdf or tiff) is created with createPDFFile='true'.
2. A document describing XML file will be generated if createXMLFile='true' is set (default). For more information see also chapter File DMS plugin, section "XML file with document information".
3. Signature image file(s) is(are) written if createSignatureImageFile='true' is configured (default).

   The image format of a captured (signware) signature can be configured with setting createSignatureImageFormat. Possible values are "png" (default), "jpg" (or "jpeg"), "gif" or "tiff". The resolution (in dots per inch) of the written signature image is set by createSignatureImageDPI entry with default string value "300".

   Any other (non signware signature) content of a signature field (i.e. camera captured image or Click-to-Sign signature) is written as JPEG image by default.

   With resampleImageSignatures setting it is possible to create the image in another format, whereas createSignatureImageFormat setting is used here also for the definition of the image format (with 'png' as default).
4. A TIFF image copy can be created from a pdf document with createTIFFCopy='true' (default is 'false').

5. A "rdy"-file (rdy as abbreviation for ready) is created at the end of all file operations in FileDMS plugin if createRDYFile='true' (default) is configured.

Example

A document file can be requested with

```
sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createPDFFile='true'
```

File name compositions

A file name can be composed of different parts.

1. The file name prefix from plugin specific setting filenamePrefix (sdweb_config.groovy), default "" .
2. The base file name from value of MetaData entry whose key comes from metadataKeyForFilename setting, otherwise docid is used as default file name.
3. A unique timestamp (14 chars) if forceCreateUniqueFiles is set to true, default "" .
4. If document file is requested (setting createPDFFile='true', default): The document name postfix from setting documentFileDecoration, default "" .

   If xml file is requested (setting createXMLFile='true', default): The xml name postfix from setting xmlFileDecoration, default "" .

   If Signature Image file is requested (setting createSignatureImageFile='true', default): The xml name postfix from setting xmlFileDecoration, default "" .
5. The file name postfix from setting filenamePostfix, default ""

   "!!!SIGNATURE!!!" for createSignatureImageFile='true'

   "_copy" for createTIFFCopy
6. The dot character '.' .
7. The file extension, dependent from the document type for setting createPDFFile='true'.

   "xml" for setting createXMLFile='true'.

   "!!!EXTENSION!!!" for createSignatureImageFile='true'.

   "rdy" for createRDYFile='true'.

   "tiff" for createTIFFCopy='true'.

   If storeStrategy setting is set to "folder" (default) the requested files are stored in a further subfolder with the same base file name as 2. part of file name (see above).

Compendium of file name compositions

1. Document file (createPDFFile):

   ```
   filenamePrefix + baseFilename + unique_timestamp + documentFilePostfix +
   filenamePostfix + "." + fileExtension
   ```
2. XML file (createXMLFile):

   ```
   filenamePrefix + baseFilename + unique_timestamp + xmlFilePostfix +
   filenamePostfix + ".xml"
   ```
3. Signature file (createSignatureFile):

   ```
   filenamePrefix + baseFilename + unique_timestamp + xmlFilePostfix +
   filenamePostfix + "!!!SIGNATURE!!!" + "." + "!!!EXTENSION!!!"
   ```

4. Tiff document copy (createTIFFCopy):
   `filenamePrefix + baseFilename + unique_timestamp + filenamePostfix + "_copy.tiff"`
5. Ready file (createRDYFile):
   `filenamePrefix + baseFilename + unique_timestamp + filenamePostfix + ".rdy"`

- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".storeStrategy** (string): The documents are stored in separate directories having the name of the specified documentid (or the configured file name) if 'folder' is configured. With value 'flat' the documents are stored without the folder structure. Default: "folder"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".filenamePrefix** (string): File name prefix, see section "File name compositions". Default: ""
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".filenamePostfix** (string): File name postfix, see section "File name compositions". Default ""
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createRDYFile** (string): If "true" a "rdy" file is created as soon as all file operations has been finished by FileDMS plugin. Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createXMLFile** (string): If "true" an XML file with document information is generated, (see also section "XML file with document information". Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".documentFileDecoration** (string): The document file postfix for document files if createPDFFile="true", see documentFilePostfix in section "File name compositions". Default: "_document"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".xmlFileDecoration** (string): The XML name postfix for xml files if createXMLFile="true", see xmlFilePostfix in section "File name compositions". Default: "_content"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".dmsFolder** (string): Location of FileDMS output files (Note: ${sdweb.home} is value from global SDWEB_HOME environment variable). Default: "${sdweb.home}/dms/de.softpro.sdweb.plugins.impl.FileDms"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createTIFFCopy** (string): If "true" a TIFF image copy of the document is generated from a pdf document. Default: "false"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createSignatureImageFile** (string): If "true" Signature Image file(s) is(are) created from signed capture fields. Default: "false"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createSignatureImageFormat** (string): Signature image format from created signature images (if createSignatureImageFile="true"). Possible image formats are "png" (default), "jpg" (or "jpeg"), "gif" or "tiff". Default: "png"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createSignatureImageDPI** (string): The resolution (in dots per Inch) of generated Signature images (if createSignatureImageFile="true"). Default: "300"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".storeBiometricDataInXML** (string): The Base64 encoded signature (signware format) is included as value of the biosig element in the XML file storeBiometricDataInXML is set to 'true' (default). See also section "XML file with document information". Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".fileLockTimeout** (integer): The maximum lock time in minutes for a document before a lock is released. This is relevant if document lock support

is enabled for documents which are loaded from DMS. Note: The lock from a document is released in FileDMS after the 'rdy' is written. Default: 10

- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createPDFFile** (string): The (archived) PDF file is written to FileDMS output folder if set to "true". Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".resampleImageSignatures** (string): If the content of a capture field is an image which was grapped by a camera or as a Click-to-Sign signature (also not a signware signature) the image is written (createSignatureImageFile="true") in JPEG format. The format can be changed if resampleImageSignatures="true" is set. The changed format can be specified via createSignatureImageFormat setting.Default: "true"

XML file with document information

Important There is no DTD or XML-schema file for the contents of the created XML file available. The structure and content of this file may change over time. It is not recommended to use this file as interface for other systems. If such an XML file is required for integration with other systems, development of a custom DMS plugin should be considered instead.

A generated XML file contains information about the archived document within the sdweb element.

The subordinated document element has the document id as attribute.

The document element has the following subelements:

1. metadata with entry and subordinated key/value elements
2. formfields with field elements, containing the name, type and the value of the field
3. takensignatures contains a biosig entry for each signed capture field with the attributes timestamp, name and signed='true' flag. The Base64 encoded signature (signware format) is included as value of the biosig element if storeBiometricDataInXML is set to 'true' (default).
4. allsignatures contains similar to takensignatures also any capture field biosig elements with the attributes timestamp, name and signed flag. The value contains also the encoded signware signature if available and configured.
5. logicalfieldnames contains the possible mapping from any logical field names to real field names. Each included logicalfieldname has the logical id (name) as attribute and the real field name as subelement (realfieldname) also with an id attribute (both id's could be identical).
6. realfieldnames contains the same mapping but the other way round. The higher level element is the realfieldname and the subelement is the logicalfieldname.

Example extract of a generated XML file from an archived TrapezaOpenJointAccounts.pdf (Kofax sample):

```
<sdweb version="1.0">
<document id="2014-05-26_09-41-59-232">
<metadata>
<entry>
<key>SIGNDOCWEB_INTERNAL_VALIDATEPLUGIN_ID</key>
<value>de.softpro.sdweb.plugins.impl.DefaultValidator</value>
</entry>
<entry>
<key>SIGNDOCWEB_INTERNAL_RESULTURL</key>
<value>http://localhost:8080/sdweb/result/index?showoptions=true</value>
</entry>
```

```
<entry>
<key>SIGNDOCWEB_REQUIRED_form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_ASV_permitted_SPFID[0]</
key>
<value>false</value>
</entry>
<entry>
<key>...</key>
<value />
</entry>
....
</metadata>
<formfields>
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Date_SPFID[0]" type="TEXTFIELD" />
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_C_Street_SPFID[0]"
 type="TEXTFIELD" />
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_C_First_Name_SPFID[0]"
 type="TEXTFIELD" />
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Applicant_Signature_SPFID[0]"
 type="SIGNATURE_SIGNWARE" signed="true" />
....
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_A_Mandate_UpTo5000_SPFID[0]"
 checked="false" type="CHECKBOX" />
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Customer_Name_SPFID[0]"
 type="TEXTFIELD">Max Mustermann</field>
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_A_City_SPFID[0]"
 type="TEXTFIELD" />
<field name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Customer_Number_SPFID[0]"
 type="TEXTFIELD" />
....
</formfields>
<takensignatures>
<biosig timestamp="1401097349000"
 name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Applicant_Signature_SPFID[0]"
 signed="true">mBkRB/ULAAAxBQAAeJxtlglsVlUQ...4GYPa9/sN9rBsX4NPFfmOuH+g==</biosig>
</takensignatures>
<allsignatures>
<biosig timestamp="0"
 name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Co_Applicant_Signature_SPFID[0]"
 signed="false" />
<biosig timestamp="1401097348000" name="SDWEB_DEMO_MODE_FIELD_1" signed="true" />
<biosig timestamp="1401097349000"
 name="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Applicant_Signature_SPFID[0]"
 signed="true">mBkRB/ULAAAxBQAAeJxtlglsVlUQ...4GYPa9/sN9rBsX4NPFfmOuH+g==</biosig>
</allsignatures>
<logicalfieldnames>
<logicalfieldname id="Account_Title">
<realfieldname id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Account_Title_SPFID[0]" />
</logicalfieldname>
<logicalfieldname id="A_City">
<realfieldname id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_A_City_SPFID[0]" />
</logicalfieldname>
<logicalfieldname id="A_Mandate_UpTo10000">
<realfieldname
 id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_A_Mandate_UpTo10000_SPFID[0]" />
</logicalfieldname>
....
<logicalfieldname id="C_Mandate_Unlimited">
<realfieldname
 id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_C_Mandate_Unlimited_SPFID[0]" />
</logicalfieldname>
</logicalfieldnames>
<realfieldnames>
<realfieldname id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_Account_Title_SPFID[0]">
<logicalfieldname id="Account_Title" />
```

```
</realfieldname>
<realfieldname id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_A_City_SPFID[0]">
<logicalfieldname id="A_City" />
</realfieldname>
<realfieldname
 id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_A_Mandate_UpTo10000_SPFID[0]">
<logicalfieldname id="A_Mandate_UpTo10000" />
</realfieldname>
....
<realfieldname
 id="form1[0].SPFID_AOPN_DEMO_SPFID[0].SPFID_C_Mandate_Unlimited_SPFID[0]">
<logicalfieldname id="C_Mandate_Unlimited" />
</realfieldname>
</realfieldnames>
</document>
</sdweb>
```

SFTP DMS plugin

The SFTP DMS plugin is a SignDoc Web standard plugin that stores the documents on a remote SFTP Server.

Plugin test

To test the SFTP DMS Plugin you can use freeSSHd SFTP server:

1. Download freeSSHd http://www.freesshd.com/freeSSHd.exe.
2. Install freeSSHd with default values on machine where SignDoc Web server is running.
3. freeSSHd settings (version used 1.2.4):
   - Users/Add.../Login: sdweb/Authorization: Password stored as SHA1 hash/Password: sdweb/User can use: SFTP
   - SFTP/SFTP home path: specify path of choice (e.g. Documents\Download folder)
   - Server status/Start SSH server
4. Add settings to sdweb_config.groovy (hostname is defaulted to localhost and port 22):
   - sdweb.plugins.loadlist = ['de.softpro.sdweb.plugins.impl.SftpDms']
   - sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.user="sdweb"
   - sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.password="sdweb"
5. Restart SignDoc Web for the changes to become effective
6. Call SignDoc Web (e.g. http://localhost:8080/sdweb):
   - Select "SignDocWeb SFTP DMS" as Document Management System plugin (DMS)
   - Open document by URL, from template or via upload
   - Sign document
   - Use icon "Send document to archive"
7. Document is stored in SFTP location:
   - A subfolder with the document ID is created
   - The document is saved as PDF, RDY and XML file

Plugin setup

The plugin can be loaded and so used by adding the plugins class name to the 'loadlist' property in the Configuration file sdweb_config.groovy.

Load the plugin

Example

```
sdweb.plugins.loadlist = ['de.softpro.sdweb.plugins.impl.SftpDms']
```

Configuration

It can be configured to create different files regarding the archived document.

It is possible to request up to 5 different files by configuration in sdweb_config.groovy with prefix

```
'sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".'
```

followed by the appropriate setting.

The following settings constitutes creating of the remote files:

```
sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createRDYFile="..."
sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createSignatureImageFile="...."
sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createXMLFile="...."
sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createTIFFCopy="....."
```

The conditions for the file compositions are the same as for FileDms (with the same setting extension), see chapter File DMS plugin.

- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".storeStrategy** (string): The documents are stored in separate folders having the name of the specified documentid (or the configured file name) if 'folder' is configured. With value 'flat' the documents are stored without the folder structure. Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".storeStrategy ("folder")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".filenamePrefix** (string): File name prefix, see File DMS plugin section "File name compositions". Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".filenamePrefix ("")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".filenamePostfix** (string): File name postfix, see File DMS plugin section "File name compositions". Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".filenamePostfix ("")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createRDYFile** (string): If "true" a "rdy" file is created as soon as all file operations has been finished by SftpDms plugin. Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createRDYFile ("true")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createXMLFile** (string): If "true" an XML file with document information is generated, see also File DMS plugin section "XML file with document information". Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createXMLFile ("true")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".documentFileDecoration** (string): The document file postfix for document files if createPDFFile="true", see documentFilePostfix in File DMS plugin, section "File name compositions". Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".documentFileDecoration ("_document")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".xmlFileDecoration** (string): The XML name postfix for xml files if createXMLFile="true", see xmlFilePostfix

in File DMS plugin, section "File name compositions". Default: value from sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".xmlFileDecoration ("_content")

- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".dmsFolder** (string): Location (folder) of SftpDms output files on remote server. Default: "dms/de.softpro.sdweb.plugins.impl.SftpDms"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createTIFFCopy** (string): If "true" a TIFF image copy of the document is generated from a pdf document. Default: value of sdwebplugins."de.softpro.sdweb.plugins.impl.FileDms".createTIFFCopy ("false")
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createSignatureImageFile** (string): If "true" Signature Image file(s) is(are) created from signed capture fields. Default: "false"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createSignatureImageFormat** (string): Signature image format from created signature images (if createSignatureImageFile="true"). Possible image formats are "png" (default), "jpg" (or "jpeg"), "gif" or "tiff". Default: "png"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createSignatureImageDPI** (string): The resolution (in dots per Inch) of generated Signature images (if createSignatureImageFile="true"). Default: "300"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".storeBiometricDataInXML** (string): The Base64 encoded signature (signware format) is included as value of the biosig element in the XML file storeBiometricDataInXML is set to 'true' (default). See also XML file with document information. Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".strictHostChecking** (string): The default value "no" allows an explicit connection to any host and suppresses a strict inspection of the Host Keys. New Host Keys are always accepted. With "yes" new Host Keys will be never accepted automatically, they must be entered in the sshostkeys file manually. Default: "no"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".createPDFFile** (string): The (archived) PDF file is written to SftpDms output folder if set to "true". Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".resampleImageSignatures** (string): If the content of a capture field is an image which was grabbed by a camera or as a Click-to-Sign signature (also not a signware signature) the image is written (createSignatureImageFile="true") in JPEG format. The format can be changed if resampleImageSignatures="true" is set. The changed format can be specified via createSignatureImageFormat setting. Default: "true"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".serverAddress** (string): Host name of the remote SFTP server Default: "localhost"
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".serverPort** (integer): Port number for connection to remote SFTP server. Default value 22 is the standard Secure Shell (SSH) port number for encrypted file transmission. Default: 22
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".user** (string): User name for remote SFTP connection Default: ""
- **sdwebplugins."de.softpro.sdweb.plugins.impl.SftpDms".password** (string): Password for remote SFTP connection Default: ""

Servlet DMS plugin

The Servlet DMS plugin is a SignDoc Web standard plugin that pushes the documents via HTTPS or HTTP POST request on a remote server.

Only predefined URLs are valid as target.

Plugin setup

The plugin can be loaded and used by adding the plugins class name to the 'loadlist' property in the Configuration file sdweb_config.groovy.

Load the plugin

Example

```
sdweb.plugins.loadlist = ['de.softpro.sdweb.plugins.impl.ServletDms']
```

Interface description for receiving servlet

The following servlet parameters are used when submitting information to the receiver:

- **docid** (string): The document id
- **docfile** (base64 encoded binary data): The binary document (i.e. pdf or tiff document)
- **contents** (base64 encoded UTF-8 string): With XML data of the document contents
- **tiffcopy** (base64 encoded binary data): The document as a tiff copy (without digital signatures)

Interface description for storing servlet

The following servlet parameters are used when storing the document to the configured servlet url.

- **docid** (string): The document id
- **docfile** (base64 encoded binary data): The binary document (i.e. pdf or tiff document)
- **xmlfile** (base64 encoded UTF-8 string): With XML data of the document contents. An xml structure (in utf-8) with document contents and metadata (can be disabled with sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createXMLFile="false")

Configuration

The plugin is configured in sdweb_config.groovy with the following properties.

Mandatory configuration

```
sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.target_urls=["url_1",
"url_2", ...]
```

This array of Strings defines the valid URLs where the documents can be pushed to.

A single URL is selected by the simple servlet parameter:

```
tui=<1_based_index>
```

Example load statement using the tui parameter:

```
https://<server>/sdweb/load/doc?
template=my_template.pdf&dmsid=de.softpro.sdweb.plugins.impl.ServletDms&tui=1
```

Optional configuration

```
sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.use_all_targets=true
```

Default value: false

The documents will be pushed to all targets. The tui servlet parameter is optional in this case.

```
sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createXMLFile
```

Default value: true

On store the document the parameter contents can be disabled by setting the above parameter to "false". This will reduce the size of the store document request.

BasicAuthenticator plugin

The BasicAuthenticator plugin is available since SignDoc Web 4.0.

The IAuthenticate interface protects, on an application bases, essential entry points of the server, especially the load servlets that open documents. The BasicAuthenticator implements this plugin interface.

General description

The BasicAuthenticator plugin provides a pragmatic protection of the server that is based on a shared secret and two public parameters.

The client portal calculates a hash value of the three parameters and sends this hash together with the other parameter in the load request.

The server will do the same calculation using the two public parameters that come with the request and the shared secret that is not part of the request.

If the two hash values match, the request is considered to be valid.

Configuration

To enable the Basic authenticator two settings have to be done in sdweb_config.groovy file.

```
// Set the BasicAuthenticator as Authenticate pluign
sdweb.authenticate.pluginid="de.softpro.sdweb.plugins.impl.BasicAuthenticator"
// Enable the BasicAuthenticator
sdwebplugins.de.softpro.sdweb.plugins.impl.BasicAuthenticator.enabled=true
// OPTIONAL: Set the location of the file containing the shared secret
//sdwebplugins.de.softpro.sdweb.plugins.impl.BasicAuthenticator.keyfile="/some/path/to/
my_keyfile.txt"
```

The shared secret is stored in a file. The location of the file is defined by the setting:

```
sdwebplugins.de.softpro.sdweb.plugins.impl.BasicAuthenticator.keyfile
```

The default location is:

```
SDWEB_HOME/conf/auth_keyfile.txt
```

Example

```
auth_keyfile.txt
934579hgih3=9sjfhosdfk/%&&%j93
```

Servlet parameters

- auth (alternative: key)
  The hash value calculated by the caller
- docid (alternative: referenceId)
  The document id to make the hash dependent on the document
- docts (alternative: timestamp)
  A timestamp to give the hash a random component (format does not matter)

Calculation of auth parameter

Pseudo code

```
BASE64(SHA-1((docid + docts + secretKey).getBytes("ISO-8859-1")))
```

Example Java code

```
public class AuthCalc {

public static void main(String[] args) throws Exception {
// the key file is configured by the configuration setting...
//
 sdwebplugins.de.softpro.sdweb.plugins.impl.BasicAuthenticator.keyfile="PATH_TO_A_FILE"
String defaultKeyFileName = System.getenv("SDWEB_HOME") + "/conf/auth_keyfile.txt";
byte[] data = FileUtils.readFileToByteArray(new File(defaultKeyFileName));
String secretKey = new String(data, "UTF-8");
String docid = "my_document_id_123";
String docts = "" + System.currentTimeMillis();
String auth = calulateHashExample(docid, docts, secretKey);

// print a sample request form
System.out.println("<html>");
System.out.println("<form action=\"http://localhost:8080/sdweb/load/doc\" method=\"post
\">");
System.out.println("docid: <input name=\"docid\" value=\"" + docid + "\">");
System.out.println("docts: <input name=\"docts\" value=\"" + docts + "\">");
System.out.println("auth: <input name=\"auth\" value=\"" + auth + "\">");
System.out.println("template: <input name=\"template\" value=
\"TrapezaOpenJointAccounts.pdf\">");
System.out.println("<input type=submit>");
System.out.println("</form>");
System.out.println("</body></html>");
}

private static String calulateHashExample(String docid, String docts, String secretKey)
 throws UnsupportedEncodingException, NoSuchAlgorithmException {
if (docts == null) throw new IllegalArgumentException("docts is null or empty");
if (docid == null) throw new IllegalArgumentException("docid is null or empty");
MessageDigest messageDigest = MessageDigest.getInstance("SHA"); // SHA-1
byte[] input = (docid + docts + secretKey).getBytes("ISO-8859-1"); // Latin-1
messageDigest.update(input);
byte[] hash = messageDigest.digest();
String hashAsString = Base64.encode(hash); // class from Apache Commons
return hashAsString;
}

}
```

Example request

The example code above will create this valid request for the example auth_keyfile.txt as stated above.

```
<html><body>
   <form action="http://localhost:8080/sdweb/load/doc" method="post">
   docid: <input name="docid" value="my_document_id_123"><br>
   docts: <input name="docts" value="1392985562078"><br>
   auth: <input name="auth" value="eZgo+hOa8IP6gHd6TUhochy6EUk="><br>
   template: <input name="template" value="TrapezaOpenJointAccounts.pdf"><br>
   <input type=submit>
   </form>
</body></html>
```

Preload plugins

SimpleFilePreloader (default)

- **Pros**
  - The only requirement is a directory with read/write access for the application server process
  - No additional configuration is required, if only a single SignDoc Web instance is used
- **Cons**
  - Requires a network share in a clustered environment
  - Orphaned files, (i.e. unused preloaded documents) must be cleaned regularly from the preload directory
  - Can be “slow” (especially when using networks shares)
- **Configuration file settings**

```
[sdweb_config.groovy]
// set it as default preload implementation
sdweb.defaults.preload.plugin.impl=”de.softpro.sdweb.plugins.impl.SimpleFilePreloader”’

// Preloaded files are uploaded to this directory
// Optional config option. Default value: %SDWEB_HOME%/preloaded_docs
sdweb.preload.dir=<local_directory_or_network_sahre_with_read_write_access>
```

LocalCachePreloader

- **Pros**
  - Fastest option
  - No local files
  - No configuration is required
- **Cons**
  - Consumes JVM heap memory for every uploaded document
  - Does not work in clustered environments, works only with single instances
- **Configuration file settings**

```
[sdweb_config.groovy]
// set it as default preload implementation
sdweb.defaults.preload.plugin.impl=”de.softpro.sdweb.plugins.impl.LocalCachePreloader”’
```

MemcachedPreloader

- **Pros**
  - Fast option for clustered environments
  - Automatic cleanup of orphaned objects
  - No local files
- **Cons**
  - Requires separate memcached service
- **Configuration file settings**

```
[sdweb_config.groovy]
// set it as default preload implementation
sdweb.defaults.preload.plugin.impl=”de.softpro.sdweb.plugins.impl.MemcachedPreloader”

// expiration in seconds (Type: Integer, Default value: 600)
// preloaded document expire after the specified number of seconds
// and might be cleaned by the memcached service
// Since SignDoc Web: 5.2.0.1
sdwebplugins."de.softpro.sdweb.plugins.impl.MemcachedPreloader".expiration=600

// timeout in seconds (Type: Integer, Default value: 3)
// maximum time to wait for a memcached action. If timeout is exceeded an error is
 thrown
sdwebplugins."de.softpro.sdweb.plugins.impl.MemcachedPreloader".write_timeout_in_seconds=3

// write_to_all_nodes (Type: boolean, Default value: false)
// If set to true, a write action will be performed on all configured memcached
 nodes.
// If one write action passes, it is considered a success.
// If set to false, the write action will be considered as success after the first
 successful write
// on one of the configured nodes
// Remark: The configured nodes are iterated in the sequence as they are configured.
sdwebplugins."de.softpro.sdweb.plugins.impl.MemcachedPreloader".write_to_all_nodes
```

- **Environment variable**

```
// nodename: a simple identifier. Use only ASCII characters
// hostname: the full domain name of the memcached server
// port: optional. Port number of the service. Default value: 11211
SPEC_MEMCACHED_NODES=<nodename:hostname[:port]>[,<nodename:hostname:port>...]
```

Example

```
// single memcached instance with default port setting
SPEC_MEMCACHED_NODES=n1:mc1.example.com

// two memcached instance with an explicit port setting
SPEC_MEMCACHED_NODES=n1:mc1.example.com,n2:mc2.example.com:11211
```

Chapter 6

Configuration file sdweb_config.groovy

Parameters can fall into one of the below categories which are listed on the SignDoc Web About page for each parameter:

- RT: Changes made to the parameter will take effect immediately during RunTime (as per parameter sdweb.config.autoupdate.interval the config is read every 5 seconds by default)
- SR: For changes to take effect a ServerRestart will be required.
- SU: These parameters fall into the category Specification Undefined which means it depends on the runtime scenario how they behave with regards to a server restart being required or not.
- UV: These parameters have a User Value i.e. the default value was changed.
- DV: These parameters have their Default Value set.
- NA: These parameters are not available for configuration by the user but are merely listed for information purposes.

Find below a complete list of available configuration parameters in the same order as listed on the SignDoc Web About page.

- **gwtutils.config.directory** (string): Set location of folder where configuration files are located. Default: "${sdweb.home}/conf"
- **gwtutils.config.readerclass** (string): Set class that should be used to read the configuration. Default: "de.softpro.sdweb.gwt.server.configuration.SDConfigurationReader"
- **sdweb.about.display_help_url** (boolean): Defines if direct links to the help topics are displayed in the About page. Default: true
- **sdweb.about.include.config_infos** (boolean): Show or hide configuration section of About page. Default: true
- **sdweb.about.include.general_infos** (boolean): Show or hide general section of About page. Default: true
- **sdweb.about.include.hostname_section** (boolean): Show or hide host name section of About page. Default: false
- **sdweb.about.include.license_infos** (boolean): Show or hide license section of About page
  Default: true
- **sdweb.about.include.plugin_infos** (boolean): Show or hide plugin section of About page. Default: true
- **sdweb.about.include.problem_infos** (boolean): Show or hide problem section of About page. Default: true
- **sdweb.about.include.usersetting_infos** (boolean): Show or hide user settings section of About page. Default: true
- **sdweb.about.settings_page_url** (string): Defines the URL to the Help Topics. The page should provide HTML anchors for the configuration settings. Default: ""/help/en/adminguide/html/configuration_file_sdweb_config_groovy.htm""

- **sdweb.aboutbox.excludelist** (string list): Each entry in the configuration list of the About page which contains in the last parameter of a key such an exclude string is marked as a password. The value of a password is not printed, it is masked with the value xxxxx. Default: ["password", "configlink_proapp"]

  Example: If sdweb.aboutbox.excludelist contains the string "password", the value for configuration key sdweb.certificate.store.pkcs12.password is displayed in the about page as "xxxxx".
- **sdweb.action.allow.clearsignature** (boolean): Allow the removal of a signature within a signature field after it has been signed. Default: true
- **sdweb.action.allow.deletesignature** (boolean): Specifies if it is allowed to delete a signature or not. Default: true
- **sdweb.action.allow.imagesignature** (boolean): Allow the capturing of an image in a document's image field. Default: true
- **sdweb.add_custom_http_header_entries** (boolean): Enabling the custom http header option enables the user to set arbitrary information in the http responses of SignDoc Web. Supported http header types are: String, Date and Integer values. To be used together with sdweb.http.servlet_response_header.list.xxx. Default: false

  Example

  sdweb.add_custom_http_header_entries=true

  sdweb.http.servlet_response_header.list.string=["string_entry_1=1234", "string_entry_2=5678"]

  sdweb.http.servlet_response_header.list.date=["time config read=" + java.lang.System.currentTimeMillis()]

  sdweb.http.servlet_response_header.list.integer=["a_custom_number=" + 123456]
- **sdweb.audittrail.defaults.did** (string): Default value that is used in the AuditTrail, if the document id (docid parameter) is not available for an AuditTrail statement. Default: ""
- **sdweb.audittrail.defaults.tid** (string): Default value that is used in the AuditTrail, if the transaction id (tid parameter) is not available for an AuditTrail statement. Default: ""
- **sdweb.audittrail.defaults.uid** (string): Default value that is used in the AuditTrail, if the user id (uid parameter) is not available for an AuditTrail statement. Default: ""
- **sdweb.audittrail.enabled** (boolean): Enables or disables the audit trail. Default: true
- **sdweb.audittrail.locale** (string): Parameter to set the locale that the audit trail will be written in. Default: "en"
- **sdweb.audittrail.log.field_changes_as_image** (boolean): Determines whether field changes such as text entry in a document will be recorded not only in a protocol style report but also with a screenshot of the area that has changed. Default: true
- **sdweb.audittrail.log.signature_image.after_digsig**(boolean): Determines whether the signature and its surrounding area in the document will be recorded in the audit as an image. Default: false
- **sdweb.audittrail.log.signature_image.before_digsig** (boolean): Determines whether the signature image will be recorded on its own before being placed in the document´s signature field. Default: true
- **sdweb.audittrail.plugin.impl** (string): Determines the plugin which is used for audit purposes. Default: "de.softpro.sdweb.plugins.impl.SimpleAuditLog"
- **sdweb.audittrail.ressource.xls.default** (string): Determines the eXtensible Stylesheet Language file which is used for the displaying of the Audit Trail. Default: "de/softpro/signdoc/audittrail/at2html_plain.xsl"
- **sdweb.authenticate.pluginid** (string): Can be used to set an Authentication Plugin in SignDoc Web. Default: ""

- **sdweb.browserplugin.activex.clsid** (string): CLSID of Browser Plugin. This setting is only relevant when using old browser plugin in Internet. Default: "CLSID:59478393-D7F4-40aa-BDB6-2C52A58E7C61"Explorer
- **sdweb.browserplugin.activex.minversion** (string): Specifies the minimal version of the ActiveX Browser Plugin required. If lower it needs to be upgraded. This setting is only relevant when using old browser plugin in Internet Explorer. Default: "1,0,6,5"
- **sdweb.browserplugin.netscape.id** (string): Name of Browser Plugin in log files. Default: "SP-PDF-Plugin"
- **sdweb.browserplugin.npapi.mimetype** (string): Mime type of SOFTPRO Browser Plugin. Default: "application/softpro-pdf-plugin"
- **sdweb.browserplugin.padclass** (string): The setting sdweb.browserplugin.padclass can be used to fix a specific search sequence for capture devices in SignDoc Web. For more details see *SignDoc Web Administrator's Guide*, chapter "Configure server", section "Description of sdweb.browserplugin.padclass". Default: ""
- **sdweb.browserplugin.padconfiguration**(string): This setting can be used for additional tablet configuration. Configuration data equals the options as described in tablet.ini, depending on the detected tablet model. For more details see *SignDoc Web Administrator's Guide*, chapter "Configure server, section "Description of sdweb.browserplugin.padclass". Default: ""
- **sdweb.browserplugin.padconfiguration_remotetablet.setlanguage** (boolean): Defines, if a remote tablet i.e. Sign2Phone should use the sessions language setting. Default: true
- **sdweb.browserplugin.querypad.always** (boolean): The default setting "true" makes sure that the 'QueryPad' function call to the browser plugin is invoked at each signature capturing in the browser client. This enables the user to select various capture devices for signature capture within a document. Default: true
- **sdweb.c2s.defaults.signaturerenderer.impl** (string): Default rendering engine for Click-to-Sign signatures. Default: "de.softpro.sdweb.plugins.impl.c2s.DefaultC2SSignatureRenderer"
- **sdweb.c2s.signaturereimage.dpi** (integer): Determines the resolution of the Click-to-Sign signature image. Default: 300
- **sdweb.capture.html5_desktop** (string): Define instruction for client to offer HTML5 capturing on the desktop

  Possible values:

  "auto" - no explicit presetting, client has to decide (not yet supported)

  "force" - the client is only allowed to offer HTML5 capturing via JavaScript

  "deny" - the client must not offer HTML5 capturing (default for now will be replaced by auto later)

  For this parameter to become effective sdweb.gui.desktop.impl="showjsmobile" needs to be set.

  Default: "deny"
- **sdweb.capture.html5_mobile** (string): Define instruction for client to offer HTML5 capturing on mobile devices

  Possible values:

  "auto" - no explicit presetting, client has to decide (not yet supported)

  "force" - the client is only allowed to offer HTML5 capturing via JavaScript

  "deny" - the client must not offer HTML5 capturing (default for now will be replaced by auto later)

  Default: "deny"
- **sdweb.capture.subtype.choice** (string list): If a signature field was inserted via command interface as capture field the user is enabled to select from a capture method choice list if he clicks on the field.

The possible choice list default entries are 'signature', Image_capture' and 'c2s' (for Click-to-Sign). These default values can be overwritten for each field via command interface. This choice list is also available for signature fields without a specific subtype definition ('signature', 'image_capture' or 'c2s') if sdweb.digsig.unspecified.allow.subtype.choice=true is configured. Default: ['signature', 'image_capture', 'c2s']

- **sdweb.capture.tabletpc.background_image.height** (integer): Sets height of capture dialog when using Tablet PC with browser plugin. Default: 480
- **sdweb.capture.tabletpc.background_image.width** (string): Sets width of capture dialog when using Tablet PC with browser plugin. Default: Default: 800
- **sdweb.certificate.store.pkcs12.file** (string): Path to default PKCS12 Certificate which is used to digitally sign the signatures. Default: "${sdweb.home}/conf/cert_store.p12"
- **sdweb.certificate.store.pkcs12.password** (string): Password for default PKCS12 Certificate which is used to digitally sign the signatures. Default: "secret"
- **sdweb.cmd.allow.update.readonly.editfields** (boolean): Allow the updating of read-only document fields via a SignDoc Web command. Default: true
- **sdweb.cmd.error.throwException** (boolean): Throws an Exception if a passed command is invalid, otherwise (false) only an error (or info) is logged and processing continues without the (invalid) command. Default: true
- **sdweb.command.list_separator** (string): Separator sign for commands. Default: "\|"
- **sdweb.config.autoupdate.enabled** (boolean): Automatic update of the configuration. If set to true the configuration will be update regularly as per parameter sdweb.config.autoupdate.interval. Default: true
- **sdweb.config.autoupdate.interval** (integer): The update interval for automatic configuration updates in milliseconds. The server periodically check for changes of the configuration files using this interval. The check is executed independently and asynchronously of other processes/events of the server. I.e. the maximum time that passes until a config change is recognized, is the value of this setting. Default: 5000
- **sdweb.custom.help** (string): URL to customized user guide. Default: ""
- **sdweb.custom.help_admin** (string): URL to customized administration guide. Default: ""
- **sdweb.custom.help_dev**(string): URL to customized developer guide. Default: ""
- **sdweb.debug.performancewatch.name.'10-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "LOADER-PHASE"
- **sdweb.debug.performancewatch.name.'11-00'** (string): can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "LP-CL
- **sdweb.debug.performancewatch.name.'12-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "LP-DL"
- **sdweb.debug.performancewatch.name.'20-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PREPARE-PHASE"
- **sdweb.debug.performancewatch.name.'21-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-CORE-PREFILL"

- **sdweb.debug.performancewatch.name.'22-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-PLUGIN-COMPLETE"
- **sdweb.debug.performancewatch.name.'22-01'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-PLUGIN-EXT"
- **sdweb.debug.performancewatch.name.'22-02'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-PLUGIN-STD"
- **sdweb.debug.performancewatch.name.'22-03'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-PLUGIN-POPULATE"
- **sdweb.debug.performancewatch.name.'22-04'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-MISC
- **sdweb.debug.performancewatch.name.'23-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-INIT-DOCUMENT"
- **sdweb.debug.performancewatch.name.'24-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-OPTIMIZE-SIZE"
- **sdweb.debug.performancewatch.name.'25-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-READONLY-MODE"
- **sdweb.debug.performancewatch.name.'26-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PP-CREATE-KEY"
- **sdweb.debug.performancewatch.name.'30-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "CONVERSION-PHASE"
- **sdweb.debug.performancewatch.name.'40-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "INTERACTIVE-PHASE"
- **sdweb.debug.performancewatch.name.'41-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP_PREP_SIG_PLUGIN"
- **sdweb.debug.performancewatch.name.'42-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP_PREP_IMG_PLUGIN"
- **sdweb.debug.performancewatch.name.'43-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP-ADD-SIG"
- **sdweb.debug.performancewatch.name.'44-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP-DEL-SIG"

- **sdweb.debug.performancewatch.name.'45-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP-FIELD-UPDATE"
- **sdweb.debug.performancewatch.name.'45-01'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP-FIELD-VALIDATE-UPDATE"
- **sdweb.debug.performancewatch.name.'46-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP-VALIDATE"
- **sdweb.debug.performancewatch.name.'47-00'** string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "IP_ADD_IMAGE"
- **sdweb.debug.performancewatch.name.'50-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "FINALIZE-PHASE"
- **sdweb.debug.performancewatch.name.'51-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "FP-VALIDATE"
- **sdweb.debug.performancewatch.name.'52-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "FP-STORE-DMS
- **sdweb.debug.performancewatch.name.'90-00'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-LAST-RENDERED-PAGE"
- **sdweb.debug.performancewatch.name.'90-01'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-RENDER-PRINT"
- **sdweb.debug.performancewatch.name.'90-02'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-DOWNLOAD-DOCUMENT"
- **sdweb.debug.performancewatch.name.'90-03'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "PRINT_VIEW"
- **sdweb.debug.performancewatch.name.'90-04'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-SERVERDONE"
- **sdweb.debug.performancewatch.name.'90-05'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-FIRSTPAGE-SENT"
- **sdweb.debug.performancewatch.name.'99-03'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-3"
- **sdweb.debug.performancewatch.name.'99-04'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-4"

- **sdweb.debug.performancewatch.name.'99-05'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-5"
- **sdweb.debug.performancewatch.name.'99-06'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-6"
- **sdweb.debug.performancewatch.name.'99-07'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-7"
- **sdweb.debug.performancewatch.name.'99-08'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-8"
- **sdweb.debug.performancewatch.name.'99-09'** (string): Can be used to change the name of the respective phase as it appears in the log files. Specify with single quotes: sdweb.debug.performancewatch.name.'xx-xx'="XYZ". Default: "MISC-9"
- **sdweb.defaults.customloader.plugin.impl** (string): Name of the default custom loader plugin to be used. Default: ""
- **sdweb.defaults.defaultpublickey.name** (string): The public key file name which is used to encrypt the biometric data of a signature. Default: "0001-public.key"
- **sdweb.defaults.imagecapturefield.image.halignment** (integer): The horizontal alignment (ha_left, ha_center, or ha_right) of the image (in the appearance stream of PDF documents) of an image capture field. Possible values: 0 - ha_left, 1 - ha_center, 2 - ha_right. Default: 1
- **sdweb.defaults.imagecapturefield.image.margins** (integer): Margins in millimeters around a captured image within the rectangle. Default: 0
- **sdweb.defaults.imagecapturefield.image.valignment** (integer): The vertical alignment (va_top, va_center, or va_bottom) of the image (in the appearance stream of PDF documents) of an image capture field. Possible values: 0 - va_top, 1 - va_center, 2 - va_bottom. Default: 1
- **sdweb.defaults.imagecapturefield.text.position** (integer): Position of the text in an image capture field. Possible values: 0 - tp_overlay (Text and image are independent and overlap, text is painted on image), 1 - tp_below (Text is put below the image, the image is scaled to fit), 2 - tp_underlay (Text and image are independent and overlap, text is painted under image). Default: 1
- **sdweb.defaults.preload.plugin.impl** (string): The Preload Plugin which is used per default. Default: "de.softpro.sdweb.plugins.impl.SimpleFilePreloader"
- **sdweb.defaults.prepare.plugin.impl** (string): The Prepare Plugin which is used per default. Default: ""
- **sdweb.defaults.signature.date.format** (string): Format in which the timestamp of the signature is displayed if sdweb.signature.display.signtime=true. Pattern follows the rules defined in Java's DateFormat class: http://docs.oracle.com/javase/7/docs/api/java/text/DateFormat.html. Default: "yyyy-MM-dd HH:mm"
- **sdweb.defaults.signature.date.locale** (string): Locale of signature timestamp which is displayed if sdweb.signature.display.signtime=true. Default: "en"
- **sdweb.defaults.signature.displaytext** (string): The signature field will show the specified signer name if sdweb.signature.display.signer=true.This setting must be assigned a String value! sdweb.defaults.signature.displaytext= is invalid!, sdweb.defaults.signature.displaytext="" is valid. Default: "SignDoc"
- **sdweb.defaults.signature.logo** (string): Defines a free text that can be displayed in a C2S signature. Default: "E-SIGN WITH KOFAX"

- **sdweb.defaults.signature.penwidth** (integer): Stroke width of signature image in document independent of signer´s signature. Default: 750
- **sdweb.defaults.signaturearchive.impl** (string): The Signature Archive Plugin which is used per default. Default: ""
- **sdweb.defaults.signaturefield.image.halignment** (integer): The horizontal alignment (ha_top, ha_center, or ha_bottom) of the image (in the appearance stream of PDF documents) of signature field. Possible values: 0 - ha_top, 1 - ha_center, 2 - ha_bottom. Default: 1
- **sdweb.defaults.signaturefield.image.margins** (integer): Margins in millimeters around the signature image within the rectangle. Default: 1
- **sdweb.defaults.signaturefield.image.valignment** (integer):The vertical alignment (va_top, va_center, or va_bottom) of the image (in the appearance stream of PDF documents) of a signature field. Possible values: 0 - va_top, 1 - va_center, 2 - va_bottom. Default: 1
- **sdweb.defaults.signaturefield.text.color** (string):

  Color of the text in the signature field as Hexadecimal RGB value. See http://www.w3schools.com/html/html_colors.asp. It is applicable if at least one of following settings are true:

  sdweb.signature.display.signtime=true

  sdweb.signature.display.signer=true

  Default: "808080"
- **sdweb.defaults.signaturefield.text.font.name** (string): Font of the text in the signature field. It is applicable if at least one of following settings is true:

  sdweb.signature.display.signtime=true

  sdweb.signature.display.signer=true

  Default: "Helvetica"
- **sdweb.defaults.signaturefield.text.font_size** (integer): Font size of the text in the signature field. It is applicable if at least one of following settings is true:

  sdweb.signature.display.signtime=true

  sdweb.signature.display.signer=true

  Default: 0
- **sdweb.defaults.signaturefield.text.halignment** (integer): The horizontal alignment (ha_top, ha_center, or ha_bottom) of the text (in the appearance stream of PDF documents) of signature field. Possible values: 0 - ha_top, 1 - ha_center, 2 - ha_bottom. Default: 1
- **sdweb.defaults.signaturefield.text.margins.horizontal** (integer): Horizontal margins of text in a signature field in millimeters. Default: 3
- **sdweb.defaults.signaturefield.text.position** (integer): Position of the text in a signature field. Possible values: 0 - tp_overlay (text and image are independent and overlap, text is painted on image), 1 - tp_below (text is put below the image, the image is scaled to fit), 2 - tp_underlay (text and image are independent and overlap, text is painted under image). Default: 1
- **sdweb.defaults.signaturefield.text.valignment** (integer):Vertical alignment of the text within a signature field. Possible values: 0 - va_top, 1 - va_center, 2 - va_bottom. Default: 2
- **sdweb.demo_mode.enabled** (boolean): In general a SOFTPRO demo logo is inserted into the upper left corner of a document page if the user captures a signature with the SOFTPRO Mobile App without having the Pro-App version enabled. For demo or test purposes the demo mode can be activated for the desktop client also, to insert the demo logo after capturing by enabling this setting. Default: false
- **sdweb.digsig.unspecified.allow.subtype.choic** (boolean): Signature fields in SignDoc Web can be itemized via Command Interface as a specific subtype, like signature, image_capture or c2s (for

Click-to-Sign). In this case it is only possible to fill the signature fields with the predefined capture methods according the specified subtype. If a digital signature field was inserted in the document without SignDoc Web specific subtype definition it is possible to treat them as (also SignDoc Web specific) capture field. With setting

sdweb.digsig.unspecified.allow.subtype.choice=true

the user is enabled to select a capture method according the choice list which is specified by default with sdweb.capture.subtype.choice definition.

With

sdweb.digsig.unspecified.allow.subtype.choice=false

these unspecified signature are treated as normal signature fields without the possibility to select any other capture method (for image capturing or click to sign signatures). Default: false

- **sdweb.document.signature.tsa.config** (string): Determines the Timestamp Server Authority Server URL according to RFC 3161 which will be used for the signature timestamp. See *SignDoc Web Administrator's Guide*, topic "TSA Functionality" for further information. Default: ""
- **sdweb.document.signature.tsa.use_metadata_config** (boolean):This setting can be used to enable TSA functionality via a document's metadata. See *SignDoc Web Administrator's Guide*, topic "TSA functionality" for further information. Default: false
- **sdweb.document.text.color** (string): Changes text color of document to this Hexadecimal RGB value. See http://www.w3schools.com/html/html_colors.asp. Default: "000000"
- **sdweb.document.text.opacity** (string): Changes opacity of document's text. Default: "1.0"
- **sdweb.documentation.display.public** (boolean): The product documentation is by default available from everywhere. If you change the setting to "false" the help can only be accessed by local users via http://localhost:8080/sdweb/help. Default: true
- **sdweb.excluded.actions** (string list): Contains the context relative links which are disabled in SignDoc Web. A status code 404 (not found) will return if anybody tries to call one of the included links. Default: ["test/", "tools/", "status/"]
- **sdweb.external_server_url** (string): The server can be configured to use a fixed URL for all absolute links. This makes it possible to use SignDoc Web behind a proxy. Default: ""

  Example

  sdweb.external_server_url="http://MY_PROXYSERVER/sdweb"
- **sdweb.external_server_url_list** (string list): This is a highly specialized setting which can be used to run SignDoc Web behind multiple Proxy Servers. It is not recommended to make use of this setting in a production environment. Default: []
- **sdweb.flatten_document.download** (boolean): If set to true this parameter will remove the possibility to edit the fields after downloading. The fields will not show as editable anymore. Default: false
- **sdweb.font.configfile**(string): Path to the default font configuration file. Default: "${sdweb.home}/fonts \SPFontConfig.xml"
- **sdweb.gui.desktop.impl** (string): Show different types of GUI. For Remote Interface it is required to use the mobile GUI. Possible values: "showjs" - standard GUI, "showjsmobile" - mobile GUI. Default: "showjs"
- **sdweb.gui.impl** (boolean): Checks if the text entered in the document fields has Latin characters. If non-Latin characters are entered a warning will be displayed and the entry of the characters refused. Default: true
- **sdweb.gui.input.text.check.islatin** (boolean): Checks if the text entered in the document fields has Latin characters. If non-Latin characters are entered a warning will be displayed and the entry of the characters refused. Default: true

- **sdweb.gui.input.text.check.pattern** (boolean): The verification of text input is done in SignDoc Web client according the validation pattern (regular expression) which can be defined via Textfield parameter (validpattern) in command. For security reason it is advisable to check the entered text also on server side. This (additional) check is performed on server side if sdweb.gui.input.text.check.pattern=true is set. Default: true
- **sdweb.gui.mobile.impl** (string): Default GUI that is used on mobile clients. Default: "showjsmobile"
- **sdweb.gui.mobile.ios.activecaching.enable** (boolean): SignDoc Web mobile App under iOS needs a specifc caching behavior for effective handling with data from the server which is activated by default. It can be disabled by setting sdweb.gui.mobile.ios.activecaching.enable = false, but it is not recommended. Default: true
- **sdweb.gui.render.format** (string): Document page image format displayed in desktop browser. Supported image formats are "gif", "png", "bmp", "tiff" and "jpeg". See also *SignDoc Web Administrator's Guide*, chapter "Reduce network data". Default: "png"
- **sdweb.gui.render.format_mobile** (string): Document page image format for (known) mobile devices. Supported image formats are "gif", "png", "bmp", and "jpeg". See also *SignDoc Web Administrator's Guide*, chapter "Reduce network data".
- **sdweb.gui.render.index_color_model** (string): Color Model to be used for rendering of documents. See also SignDoc Web - Administrator's Guide, chapter Reduce Network Data for more details on color models. Default: ""
- **sdweb.gui.render.index_color_model_desktop**(string): Color Model to be used for rendering of documents on the Desktop client. See also SignDoc Web - Administrator's Guide, chapter Reduce Network Data for more details on color models. Default: ""
- **sdweb.gui.render.index_color_model_mobile** (string): Color Model to be used for rendering of documents on the Mobile client. See also See also SignDoc Web - Administrator's Guide, chapter Reduce Network Data for more details on color models. Default: ""
- **sdweb.gui.render.jpeg.quality** (integer): Adjusts jpeg quality (range: 0-100) for rendered document pages if jpeg image format is configured. It sets the compression quality to a value between 0 and 100. For lossy compression schemes, the compression quality should control the tradeoff between file size and image quality (for example, by choosing quantization tables when writing JPEG images). For lossless schemes, the compression quality may be used to control the tradeoff between file size and time taken to perform the compression. A compression quality setting of 0 is most generically interpreted as "high compression is important", while a setting of 100 is most generically interpreted as "high image quality is important". Default: 60
- **sdweb.gui.taborder** (string): Set tab order in GUI. Possible values: "appearance" and "pdf". Value "pdf" inherits tab order of the pdf document. The value "appearance" evaluates the order according to the sequence of the fields appearance from top-down to left-right. Default: "appearance"
- **sdweb.gui.ie.device_connector.enabled** (boolean): If set to true the device connector is used for capturing in Internet Explorer. Per default it is set to false and the 'old' ActiveX browser plugin is used in Internet Explorer. Note that all other browsers use the device connector for capturing. Default: false
- **sdweb.http.servlet_response_header.list.date** (string): This option allows the user to set arbitrary Date information in the http responses of SignDoc Web. To be used together with sdweb.dd_custom_http_header_entries=true. See example there. Default: []
- **sdweb.http.servlet_response_header.list.integer** (string list): This option allows the user to set arbitrary Integer information in the http responses of SignDoc Web. To be used together with sdweb.dd_custom_http_header_entries=true. See example there. Default: []

- **sdweb.http.servlet_response_header.list.string** (string list): This option allows the user to set arbitrary String information in the http responses of SignDoc Web. To be used together with sdweb.dd_custom_http_header_entries=true. See example there. Default: []
- **sdweb.image_capture.dynamic_dimension_calculation** (boolean): Alternative to fix image capture size is the dynamic calculation of the image according the field size and configured resolution (see sdweb.image_capture.dynamic_dimension_calculation_resolution). The minimum image height is the value of the setting sdweb.image_capture_height. The minimum image width is the value of the setting sdweb.image_capture_width. If "false" is set then the fixed default capture image sizes are used (maximum size, see settings: sdweb.image_capture_height and sdweb.image_capture_width). Default: true
- **sdweb.image_capture.dynamic_dimension_calculation_resolution** (integer): Default resolution as base for the calculation of the image capture size dependent from the field size. Default: 96
- **sdweb.image_capture_height** (integer): Capture image setting: Height of captured image, mm for Scanner, pixel for camera. Default: 240
- **sdweb.image_capture_width** (integer): Capture image setting: Width of captured image, mm for Scanner, pixel for camera. Default: 320
- **sdweb.install.browser_plugin.show_close_link** (boolean): Display of close link in capture dialog if the necessary browser plugin is not installed. The settings sdweb.install.browser_plugin.display_links.ie and sdweb.install.browser_plugin.display_links.npapi are also considered. If one of them is set to "false", the close link is not displayed for the appropriate browser type, independent of the value for sdweb.install.browser_plugin.show_close_link. If sdweb.install.browser_plugin.display_links.npapi is set to "false" no link is displayed in the capture dialog for Firefox and Chrome browser. If sdweb.install.browser_plugin.display_links.ie is set to "false" no link is displayed in the capture dialog for Internet Explorer. This setting is only relevant when using old browser plugin in Internet Explorer. Default: true
- **sdweb.install.browser_plugin.show_url** (boolean): Display of link to plugin test page in capture dialog if the needed browser plugin is not installed. The settings sdweb.install.browser_plugin.display_links.ie and sdweb.install.browser_plugin.display_links.npapi are also considered. If one them is set to "false", the url link is not displayed for the appropriate browser type, independent of the value for sdweb.install.browser_plugin.show_url. If sdweb.install.browser_plugin.display_links.npapi is set to "false" no link is displayed in the capture dialog for Firefox and Chrome browser. If sdweb.install.browser_plugin.display_links.ie is set to "false" no link is displayed in the capture dialog for Internet Explorer. This setting is only relevant when using old browser plugin in Internet Explorer. Default: true
- **sdweb.internal_result_page.qrcode.height** (integer): With this setting it is possible to change the height of the QR code, that is shown after finalizing a document. Default: 250
- **sdweb.internal_result_page.qrcode.width** (integer): With this setting it is possible to change the width of the QR code, that is shown after finalizing a document. Default: 250
- **sdweb.internal_result_page.showloadbydms** (boolean): With this setting it is possible to decide, whether the DMS link to the finalized document should be shown in the result page. Default: true
- **sdweb.internal_result_page.showxcb** (boolean): With this setting on true, a x-callback-url and it´s QR code are displayed in the result page after finalizing a document. Default: false
- **sdweb.load.error.fail_fast** (boolean): Per default a load error is redirected to the result page. If set to true the setting sdweb.load.error.status_code is taken into account. Default: false
- **sdweb.load.error.status_code** (integer): HTTP response code that will be used in case that a load error occurs and sdweb.load.error.fail_fast is set to true. Default: 404

- **sdweb.load_page.open_in_new_window** (boolean): With this setting it is possible to decide, if the document that is loaded via a form opens in the same browser tab or in a new one. Default: false
- **sdweb.loader.upload.max_size** (integer): This parameter specifies the file size limit for the ByUpload method. Default: 5242880
- **sdweb.loader.upload.temp_dir** (string): This parameter specifies where temporary files for the ByUpload method are to be stored. Default: "PATHTOWEBSERVER\temp"
- **sdweb.lockfields.list_separator** (string): This setting allows to change the separator sign between the field names of the fields, which should be locked after signing the signature field(s). Default: ","
- **sdweb.lockfields.list_separator_uri** (string): This setting allows to change the separator sign between the field names of the fields, which should be locked after signing the signature field(s) if used with URI syntax. Default: "\|"
- **sdweb.monitor.allow_empty_filter** (boolean): Allow the user to use enter an empty filter. Default: true
- **sdweb.pkcs7.pluginid** (string): The plugin id of the ISignPKCS7 Plugin Interface. Default: "" (empty) means that the internal signing method is used.
- **sdweb.plugins.defaultloadlist** (string list): A list of plugins which should be loaded by default. Default: [VARIOUSPLUGINS]
- **sdweb.plugins.dms.allowIdOverwriting** (boolean): The dms id (e.g. FileDms) can be set in metadata of a document (key:SIGNDOCWEB_INTERNAL_DMSPLUGIN_ID). Overwriting of a dms id for a document with a servlet parameter (key:dmsid) is this case only possible, if the configuration setting sdweb.plugins.dms.allowIdOverwriting=true is set. Default: false
- **sdweb.plugins.dms.allowLoadDocuments** (boolean): Opens a document from DMS in 'read only' mode (if set to true), which means that all fields are available but cannot be changed by the user. Default: false
- **sdweb.plugins.dms.openreadonly** (boolean): Opens a document from DMS in 'read only' mode (if set to true), which means that all fields are available but cannot be changed by the user. Default: false
- **sdweb.plugins.dms.useDocumentLocking** (boolean): Setting sdweb.plugins.dms.useDocumentLocking to true will lock a document which was loaded by DMS. This means that nobody else can load the same document from DMS during this time. A document will be unlocked earliest if the user, which locks the document, will close the document, either by archiving it again, or by closing it with cancel (or after an error). The document is automatically unlocked at the latest after 10 Minutes. Default: false
- **sdweb.plugins.global.dmsid.pluginid** (string): Defines a default dms plugin to be used in certain cases. The behaviour depends on the value of sdweb.plugins.global.dmsid.strategy

  sdweb.plugins.global.dmsid.strategy="enforce" means always use this plugin no matter, if a dms plugin is specified by parameter and sdweb.plugins.global.dmsid.strategy="fallback" means use this plugin only, if no dms plugin is specified by parameter. Default: ""
- **sdweb.plugins.global.dmsid.strategy** (string): Is only relevant for plugin (id) definition within MetaData. If default pluginid is defined and no pluginid is defined in MetaData and sdweb.plugins.global.xxxpluginidxxx.strategy="fallback" is set then default pluginid is used. The default plugin is also used if the sdweb.plugins.global.xxxpluginidxxx.strategy="enforce" is configured. Default: "fallback"
- **sdweb.plugins.global.prepareplugin id.pluginid** (string): Use this parameter to define a global Prepare plugin. Default: ""
- **sdweb.plugins.global.prepareplugin id.strategy** (string): Is only relevant for plugin (id) definition within MetaData. If default pluginid is defined and no pluginid is defined in MetaData and sdweb.plugins.global.xxxpluginidxxx.strategy="fallback" is set then default pluginid is used. The default

plugin is also used if the sdweb.plugins.global.xxxpluginidxxx.strategy="enforce" is configured. Default: "fallback"

- **sdweb.plugins.global.resultparamspluginid.pluginid** (string): Use this parameter to define a global Result Parameters plugin. Default: ""
- **sdweb.plugins.global.resultparamspluginid.strategy** (string): Is only relevant for plugin (id) definition within MetaData. If default pluginid is defined and no pluginid is defined in MetaData and sdweb.plugins.global.xxxpluginidxxx.strategy="fallback" is set then default pluginid is used. The default plugin is also used if the sdweb.plugins.global.xxxpluginidxxx.strategy="enforce" is configured. Default: "fallback"
- **sdweb.plugins.global.signaturearchiveid.pluginid** (string): Use this parameter to define a global Signature Archive plugin. Default: ""
- **sdweb.plugins.global.signaturearchiveid.strategy** (string): Is only relevant for plugin (id) definition within MetaData. If default pluginid is defined and no pluginid is defined in MetaData and sdweb.plugins.global.xxxpluginidxxx.strategy="fallback" is set then default pluginid is used. The default plugin is also used if the sdweb.plugins.global.xxxpluginidxxx.strategy="enforce" is configured. Default: "fallback"
- **sdweb.plugins.global.validatepluginid.pluginid** (string): Use this parameter to define a global Validation plugin. Default: ""
- **sdweb.plugins.global.validatepluginid.strategy** (string): Is only relevant for plugin (id) definition within MetaData. If default pluginid is defined and no pluginid is defined in MetaData and sdweb.plugins.global.xxxpluginidxxx.strategy="fallback" is set then default pluginid is used. The default plugin is also used if the sdweb.plugins.global.xxxpluginidxxx.strategy="enforce" is configured. Default: "fallback"
- **sdweb.plugins.loadlist** (string list): This parameter is used to specify the plugins which should be loaded upon SignDoc Web startup. Default: []
- **sdweb.preload.filedeletion** (string): This setting defines, when the preloaded temp files will be deleted by the server. "asap" - right after the SignDoc Document was created (as soon as possible), "envcheck" - after the gui passed all env checks and document is visible, "alap" - after successfully archiving the working document (as late as possible). Default: "envcheck"
- **sdweb.prepare.allow.docid_as_parameter** (boolean): Configuration options for overwriting already existing document id via docid parameter. Default: true
- **sdweb.prepare.allow.docid_parameter_overwrite_metadata** (boolean): Configuration options for overwriting already existing document id via docid parameter. Default: true
- **sdweb.prepare.minimizefilesize** (boolean): Tries to minimize the filesize to a minimum when preparing a document. Default: true
- **sdweb.prepare.open_readonly.flatten_threshold** (integer): A document can be opened as 'read only' if parameter openreadonly is defined (see also sdweb.plugins.dms.openreadonly). Each field in the document must be set then to readonly.

  If the document contains too many fields it needs some time to set all the fields manually to 'read only'.

  Therefore there is the possibility to define a threshold from where the document is flattened (much faster), instead of setting each field flag (to readonly).

  If sdweb.prepare.open_readonly.flatten_threshold is set to -1 every field is set to read only.

  If the threshold is equal or greater than the number of fields in the document then the document is flattened before it is displayed to the user. A flattened document means that the user can see only the

image of the document, no fields are available any more. No signatures are listed then in the signature treeview of the document.

Default: -1

- **sdweb.prepare.type.addtext** (string): With this setting it is possible to change the name of the parameter for inserting text in a document via command. Default: "addtext"
- **sdweb.prepare.type.addtextrect** (string): With this setting it is possible to change the name of the parameter for inserting a text area in a document via command. Default: "addtextrect"
- **sdweb.prepare.type.formfield** (string): With this setting it is possible to change the name of the parameter for inserting a form field in a document via command. Default: "formfield"
- **sdweb.prepare.type.metadata** (string): With this setting it is possible to change the name of the parameter that is used to add metadata to a document via command. Default: "metadata"
- **sdweb.prepare.type.removefield** (string): With this setting it is possible to change the name of the parameter that can be used to delete formfields via command. Default: "removefield"
- **sdweb.prepare.type.signature** (string): With this setting it is possible to change the name of the parameter for inserting a signature field in a document via command. Default: "signature"
- **sdweb.requester.regex.android** (string): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

  The configuration list with key sdweb.requester.regex.android contains all regular expressions for user agent strings from android clients which are not recognized correctly (as android clients) from UADetector.

  Default: ".*(?:android)\b.*"
- **sdweb.requester.regex.desktop_list** (string list): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

  The configuration list with key sdweb.requester.regex.desktop_list contains all regular expressions for user agent strings from desktop clients which are not recognized correctly (as desktop clients) from UADetector.

  Default: ['^mozilla/.*\\(compatible; msie (\\d)\\.0; .*windows nt 6\\.(\\d).*wow64.*trident/7\\.0.*\\).*$'] => IE11 under Windows 8.1 desktop
- **sdweb.requester.regex.genericmobile** (string): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

  The configuration list with key sdweb.requester.regex.genericmobile contains all regular expressions for user agent strings from generic mobile clients which are not recognized correctly (as generic mobile clients) from UADetector.

  Default: "VARIOUSDEVICES"
- **sdweb.requester.regex.ipad** (string): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized

correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

The configuration list with key sdweb.requester.regex.ipad contains all regular expressions for user agent strings from ipad clients which are not recognized correctly (as ipad clients) from UADetector.

Default: ".*\b(ipad)\b.*"

- **sdweb.requester.regex.iphone** (string): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

  The configuration list with key sdweb.requester.regex.iphone contains all regular expressions for user agent strings from iphone clients which are not recognized correctly (as iphone clients) from UADetector.

  Default: ".*\b(?:iphone)\b.*"
- **sdweb.requester.regex.ipod** (string): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

  The configuration list with key sdweb.requester.regex.ipod contains all regular expressions for user agent strings from ipod clients which are not recognized correctly (as ipod clients) from UADetector.

  Default: ".*\b(?:ipod)\b.*"
- **sdweb.requester.regex.metro** (string): The User Agent string from any client is evaluated by a 3rd party software (UADetector) Unfortunately it could happen, that any client is not (yet) recognized correctly nevertheless. For this cases it is necessary to evaluate specific user agent strings before the UADetector decides whether the requesting client is a mobile device or comes from a desktop browser.

  The configuration list with key sdweb.requester.regex.metro contains all regular expressions for user agent strings from metro clients which are not recognized correctly (as metro clients) from UADetector.

  Default: ".*\b(?:windows.nt.6\.[2-9](.*touch|.*webview/)|windows.nt.6\.[2-9].*arm)\b.*"
- **sdweb.requester.uadetector.use** (boolean): Use UADetector library for User Agent recognition. Default: true
- **sdweb.response_headerinfos** (string list): Default response headers

  SignDoc Web can put some default information in the http response headers.

  The sdweb_config.groovy option with its default values is:

  sdweb.response_headerinfos=["version", "license-mode", "license-product"]

  The following header information is available in every response by default:

  sdweb-license-product (currently SDWEB for SignDoc Web or SDS for SignDoc Service)

  sdweb-version (the version number of the server)

  sdweb-license-mode (licensed or demo)

  Example

  sdweb-license-product:SDS

  sdweb-version:5.0.104_182

  sdweb-license-mode:licensed

  Default: ["version", "license-mode", "license-product"]
- **sdweb.rest.allow.update.readonly.editfields** (boolean): Allow or deny changes of values for locked/readonly (editable) fields via REST interface. Default: true

- **sdweb.rest.text.cut.maxlength** (boolean): Text value which is updated via REST interface will be cut off (or not) up to the max length of the text field. Default: true
- **sdweb.ria.css.custom.dir** (string): With this setting it is possible to change the directory of the custom css files. Default: "css"
- **sdweb.ria.image.custom.dir** (string): With this setting it is possible to change the directory of the custom image files. Default: "resources"
- **sdweb.ria.image.custom.enabled** (boolean): With this setting it is possible to allow SignDoc Web to use custom images for buttons etc. Default: false
- **sdweb.ria.signdoc.css_list** (string list): The default CSS list for the Desktop GUIs. Default: [SignDoc.css]
- **sdweb.ria.signdoc.css_list_custom** (string list): With this setting the user has the possiblity to define a list of CSS files, that should be used for a customized Desktop GUI. Default: []
- **sdweb.ria.signdoc_mobile.android.css_list** (string list): The default CSS list for Android. Default: [SignDocMobile_Android_1.css]
- **sdweb.ria.signdoc_mobile.android.css_list_custom** (string list): With this setting the user has the possiblity to define a list of CSS files, that should be used for a customized Android Mobile GUI. Default: [SignDocMobile_Android_1.css]
- **sdweb.ria.signdoc_mobile.css_list** (string list): The default CSS list for Mobile in general. Default: [SignDocMobile_Desktop.css]
- **sdweb.ria.signdoc_mobile.css_list_custom** (string list): With this setting the user has the possiblity to define a list of CSS files, that should be used for a customized General Mobile GUI. Default: []
- **sdweb.ria.signdoc_mobile.ios.css_list** (string list): The default CSS list for iOS. Default: [SignDocMobile_IPad_1.css]
- **sdweb.ria.signdoc_mobile.ios.css_list_custom** (string list): With this setting the user has the possiblity to define a list of CSS files, that should be used for a customized iOS Mobile GUI. Default: []
- **sdweb.ria.signdoc_mobile.windows.css_list** (string list): The default CSS list for Windows Mobile. Default: [SignDocMobile_Windows.css]
- **sdweb.ria.signdoc_mobile.windows.css_list_custom** (string list): With this setting the user has the possiblity to define a list of CSS files, that should be used for a customized Windows Mobile GUI. Default: []
- **sdweb.server.event.timeout.wait** (integer): The maximum time the SignDocWeb server waits for the entry of the signature on client side after clicking on a signature field. The value is the time in milliseconds, e.g. 180000 means 180 seconds.

  After this expiration the capture dialog is closed on client side and the capture process is aborted.

  For this settings to become effective a server restart is required.

  The settings only applies to capturing via browser plugin or browser extension.

  Default: 180000
- **sdweb.server.logging.config.reload.enable** (boolean): This setting allows to enable or disable the automatic re-reading of the log settings from XjLog.xml. Default: true
- **sdweb.server.logging.config.reload.interval** (integer): This setting defines the time interval in which the log file XjLog.xml is re-read if sdweb.server.logging.config.reload.enable is set to true. Default: 5000
- **sdweb.session.keepalive** (boolean): When the GUI is open, the session will by default not expire until either the document is archived or the browser is closed and the session times out. Default: true

- **sdweb.session.keepalive_session_timeout_percent** (integer): The keepalive mechanism in the browser page triggers a request for keeping session alive after xx percent of the configured session timeout duration. Default: 90
- **sdweb.session.keepalive_session_timeout_percent_android** (integer): For Android devices: The keepalive mechanism in the browser page triggers a request for keeping session alive after xx percent of the configured session timeout duration. Default: 90
- **sdweb.session.timeout.display_information** (boolean):

  If a session timeout URL is defined (see sdweb.session.timeout.page.url) this setting determines whether parameters are appended to the URL to not.

  Note: This is only useful for Android App usage which could have problems if the redirect URL have appended query parameters.

  Default: false
- **sdweb.session.timeout.interval** (integer): Session timeout in seconds. If this parameter is not set then the value from web.xml <session-timeout>...</session-timeout> is used. Default: 60
- **sdweb.session.timeout.interval_mobile** (integer): DSession timeout for mobile devices in seconds. If this parameter is not set then the value from sdweb.session.timeout.interval is taken. Default: 3600
- **sdweb.session.timeout.interval_mobile_android** (integer): Android specific session timeout. If this parameter is set to -1 then the value from sdweb.session.timeout.interval_mobile is inherited. Default: -1
- **sdweb.session.timeout.interval_mobile_genericmobile** (integer): Session timeout for all other identified mobile devices. If this parameter is set to -1 then the value from sdweb.session.timeout.interval_mobile is inherited. Default: -1
- **sdweb.session.timeout.interval_mobile_ipad** (integer): iPad specific session timeout. If this parameter is set to -1 then the value from sdweb.session.timeout.interval_mobile is inherited. Default: -1
- **sdweb.session.timeout.page.url** (integer): With this setting it is possible to define which URL should be accessed, when a session timeout occurs. Default: -1
- **sdweb.signature.biometricdata.remove** (boolean): The biometric signature data is not stored if the parameter value is true. Without biometric signature only the appearance (image) of the signature is then available in the document. Default: false
- **sdweb.signature.color.blue** (integer): With this settings it is possible to change the amount of blue colour in the signature. Default: 0
- **sdweb.signature.color.green** (integer): With this settings it is possible to change the amount of green colour in the signature. Default: 0
- **sdweb.signature.color.red** (integer): With this settings it is possible to change the amount of red colour in the signature. Default: 0
- **sdweb.signature.display.signer** (boolean): If set to true, the signature field will show the signer name. Default: false
- **sdweb.signature.display.signtime** (boolean): If set to true, the signature field will show the signing time. Default: false
- **sdweb.signature.exif.auto_rotate** (boolean): Auto rotate images which are captured (for a signature field) with rotated (non left top) orientation. Orientation is included in exif metadata of an image (if available). The image orientation can be influenced by the orientation of the camera while the user takes a picture (e.g. with a mobile device). Default: true
- **sdweb.signature.watermark.enable** (boolean): Specifies whether a watermark image is placed behind a captured signature in the signature field. If set to true, SignDoc Web will try to add a configured

watermark behind the signature. The actual captured signature is not affected by this setting only the appearance in the document. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: false

- **sdweb.signature.watermark.frame.height** (integer): Height of the watermark frame (if the value is not negative). A negative value means that the watermark frame has the same height as the signature field. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: -1
- **sdweb.signature.watermark.frame.offset_x** (integer): Horizontal offset (in pixels) between top-left corner of the signature field and the top-left corner of the watermark frame. Positive value means top-left corner of the watermark frame is on the right of the top-left corner of the signature field. A negative value means that both offsets (x and y) are not used for frame positioning within the signature field. In this case the sdweb.signature.watermark.image.alignment setting is used. The watermark image alignment is only applicable then if width and/or height of the frame is smaller than the signature field. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: -1
- **sdweb.signature.watermark.frame.offset_y** (integer): Vertical offset (in pixels) between top-left corner of the signature field and the top-left corner of the watermark frame . Positive value means top-left corner of the watermark frame is below the top-left corner of the signature field. A negative value means that both offsets (x and y) are not used for frame positioning within the signature field. In this case the sdweb.signature.watermark.image.alignment setting is used. The watermark image alignment is only applicable then if width and/or height of the frame is smaller than the signature field. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: -1
- **sdweb.signature.watermark.frame.width** (integer): Width of the watermark frame (if the value is not negative). A negative value means that the watermark frame has the same width as the signature field. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: -1
- **sdweb.signature.watermark.image.alignment** (string): The watermark image alignment is only applicable if width and/or height of the watermark image is smaller than the watermark frame and the scale is set to "actual". If scale is set to "fit", it depends on whether the frame is longer than the fitted image (horizontal alignment possible) or taller than the fitted image (vertical alignment possible). Scale option "stretch" will not have any impact since it is stretched to fit the frame. Possible alignment settings are "top-left", "top-center", "top-right", "middle-left", "middle-center", "middle-right", "bottom-left", "bottom-center" and "bottom-right". The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: "middle-center"
- **sdweb.signature.watermark.image.opacity** (integer): Specifies the watermark image opacity in percent. The value must be between 0 and 100. 100(%) means that the watermark image is completely opaque. 0 would make the watermark invisible because it is completely transparent. This setting can be useful to show the signature more prominent compared to the watermark image which is behind the signature image. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: 100
- **sdweb.signature.watermark.image.scale** (string):

  Specifies the scaling strategy of the watermark image within the watermark frame. The watermark frame is the area where the watermark image is adjusted into. The watermark frame itself is adjusted then into the signature field, either by frame offset coordinates

(sdweb.signature.watermark.frame.offset_x and sdweb.signature.watermark.frame.offset_y) or alternatively according the alignment setting (sdweb.signature.watermark.image.alignment).

"actual" will use the actual size of the watermark image when it is positioned in the watermark frame.

"fit" will scale the image proportionately to the size of the watermark frame.

"stretch" will stretch the image disproportionately to fit the entire frame.

The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: "actual"

- **sdweb.signature.watermark.image.template** (string): The name of the watermark image which should be placed as background image for a captured signature (for the document view). Supported image formats are JPG, PNG, GIF and BMP. The setting can be overwritten by an individual and signature field specific command parameter during document loading. Default: "watermark.bmp"
- **sdweb.signature.watermark.image.template_dir** (string):

  Specifies the directory from where a watermark template (defined by sdweb.signature.watermark.image.template) is loaded. The directory specification must contain the complete path information. Subdirectories are separated by forward slashes or with double back slashes under Windows.

  Example for Windows:

  'd:/sdweb_home/wm' or 'd:\\sdweb_home\\wm'

  Note: Don't forget the drive letter under Windows!

  If you want to specify another subdirectory under SDWEB_HOME directory you can specify it in the format "${sdweb.home}/sub_dir_name"

  Example

  "${sdweb.home}/watermark"

  Default: "${sdweb.home}/watermark"
- **sdweb.signature_c2s.display.signer** (boolean): Defines if the signers name should be displayed below Click-to-Sign capture fields. Default: false
- **sdweb.signature_c2s.display.signtime** (boolean): Defines if the date should be displayed below Click-to-Sign capture fields Default: false
- **sdweb.signature_image.display.signer** (boolean): Defines if the signers name should be displayed below image capture fields. Default: false
- **sdweb.signature_image.display.signtime** (boolean): Defines if the date should be displayed below image capture fields. Default: false
- **sdweb.signature_parameters.one_time_cert.keysize** (integer): For signing a signature it is necessary to have a certificate and a private key. If no private key and no certificate is provided it is possible to create both on the fly by SignDoc SDK. A key pair for the self-signed certificate is generated. The value is the number of bits (1024 through 4096, multiple of 8). Default: 1024
- **sdweb.signerspecific.store.pkcs12.password** (string): With this setting it is possible to define a password for the signer specific certificate. Default: "secret"
- **sdweb.signing.biometric_encryption** (integer):

  Defines the biometric encryption

  0 - rsa (default, requires an public key)

  1 - fixed (symmetric encryption), not save but the biometric data can be taken out without a private key. Can be useful for document conversion in different format.

  Default: 0

- **sdweb.template.directory** (string): With this setting it is possible to change the path and name of the directory that contains the template documents. Default: "c:\sdweb_home/doctemplates"
- **sdweb.template.monitor.enabled** (boolean): If the parameter is set to true the sdweb.template.directory will be monitored for new files located there which can be used via the LoadByDMS method. Default: false
- **sdweb.usage.enable.aboutpage** (boolean): Enable or disable SDWeb "About"-page Default: true
- **sdweb.usage.enable.loadpage** (boolean):Enable or disable SDWeb "homepage". Default: true
- **sdweb.usage.startpage.url** (string): With this setting it is possible to change the SignDoc Web startpage. Default: "load/form"
- **sdweb.validate.before_archive_required_flag** (boolean): By default the required document fields are validated before archiving. When the parameter sdweb.validate.before_archive_required_flag is set to false, the required fields from the document are no longer validated before archiving. Default: true
- **sdweb.validate.before_update.fieldtypes** (string list):

  List of field types which should be validated before update action is performed. The validateFieldChange() method of Validator plugin (implementing IDocumentValidator) is called if field type which should be updated is included in the list. Supported field types are, textfield, checkbox, capture and radiobutton.

  Example

  If text fields and check boxes should be validated before update the setting must be

  sdweb.validate.before_update.fieldtypes=['textfield', 'checkbox']

  Default: []
- **sdweb.web_page_options.document_domain** (string):

  Sets the allowed script origins to interact with the SignDoc Web UI. This is especially useful when using the Remote Interface. The effect is that the JavaScript DOM property document.domain is initialized with the specified value. Specify the setting in the format "[sub.domain.tld]"

  Note: A value of "" (empty string) will disable the setting

  Default: ""
- **sdweb.ws.fault.disable_stacktrace** (boolean): By default a web service (JAX-WS) marshals the complete StackTrace of an exception, this is usually not wanted. Therefore it is set to true. Set to false if the propagation of the StackTrace is needed on client side. Default: true
- **sdweb.xjlog.config.autoupdate.enabled**
- **sdwebplugins.de.softpro.sdweb.plugins.impl.BasicAuthenticator.enabled** (boolean): If this setting is disabled (false) the logging configuration XjLog.xml is not reread during runtime but a server restart is required. Default: true
- **sdwebplugins.de.softpro.sdweb.plugins.impl.BasicAuthenticator.keyfile** (boolean): If set to true a BasicAuthentication will be performed upon accessing SignDoc Web. Parameter has to be used in conjunction with sdweb.authenticate.pluginid="de.softpro.sdweb.plugins.impl.BasicAuthenticator". For more information check the *SignDoc Web Administrator's Guide*, chapter "Standard plugins", section "BasicAuthenticator plugin". Default: false
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createRDYFile** (string): With this setting it is possible to decide, whether a RDY file should be created after finalizing a document or not. The RDY (Ready) file is created as the last file of the whole finalize process and can be used as a trigger for a follow-on process. Default: "true"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createSignatureImageDPI** (string): With this setting the resolution in DPI of the signature image which is save upon finalize of the document

can be set. This is only applicable to image formats that contain DPI information e.g. TIFF, PNG, ...The default resolution for HTML5 signatures is 96DPI and is reduced in relation to the default of 300. For example if the DPI is set to 100 the HTML5 signature resolution is also reduced to one third i.e. 32DPI. Default: "300"

- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createSignatureImageFile**(string): With this setting it is possible to decide, whether an image of the signature should be created after finalizing a document or not. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createSignatureImageFormat** (string): With this setting it is possible to define the format of the signature image which will be saved if sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createSignatureImageFile is set to true. Possible formats are "png", "gif", "bmp", "jpg", "tiff" (only supported for handwritten signatures and not for photos or Click-to-Sign signatures). Default: "png"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createTIFFCopy** (string): With this setting it is possible to decide, whether a TIFF copy of the signed document should be created. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createXMLFile** (string): With this setting it is possible to decide, whether an XML file should be created after finalizing a document. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.dmsFolder** (string): With this setting it is possible to change the path and the name of the folder where the finalized documents are stored. Default: "c:\sdweb_home/dms/de.softpro.sdweb.plugins.impl.FileDms"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.fileDecoration** (string): The value of this setting is placed in all generated filenames after the docid (also after the optional timestamp) and before the optional filepostfix. A fileDecoration value could be useful for example if several SignDoc Web servers write their files to one common (shared) directory. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.filenamePostfix** (string): With this setting it is possible to define a prefix for all the files related to the document. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.filenamePrefix** (string): With this setting it is possible to define a prefix for all the files related to the document. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.storeBiometricDataInXML** (string): With this setting it is possible to define whether the biometric signature data should be stored in the XML file or not. Only applicable if sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.createXMLFile="true". Default: "true"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.FileDms.storeStrategy** (string): With this setting it is possible to decide, if the documents in the DMS should be stored in folders or not. If the storage strategy should be just files the value has to be set to "flat". Default: "folder"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createSignatureImageDPI** (string): With this setting the resolution in DPI of the signature image which is save upon finalize of the document can be set. This is only applicable to image formats that contain DPI information e.g. TIFF, PNG,... The default resolution for HTML5 signatures is 96DPI and is reduced in relation to the default of 300. For example if the DPI is set to 100 the HTML5 signature resolution is also reduced to one third i.e. 32DPI. Default: "300"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createSignatureImageFile** (string): With this setting it is possible to decide, whether an image of the signature should be created after finalizing a document or not. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createSignatureImageFormat** (string): With this setting it is possible to define the format of the signature image which will be saved if sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createSignatureImageFile is set to true.

Possible formats are "png", "gif", "bmp", "jpg", "tiff" (only supported for handwritten signatures and not for photos or Click-to-Sign signatures). Default: "png"

- **sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createXMLFile** (string): With this setting it is possible to decide, whether an XML file should be created after finalizing a document. Default: "true"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.storeBiometricDataInXML** (string): With this setting it is possible to define whether the biometric signature data should be stored in the XML file or not. Only applicable if sdwebplugins.de.softpro.sdweb.plugins.impl.ServletDms.createXMLFile="true" Default: "true"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createRDYFile** (string): With this setting it is possible to decide, whether a RDY file should be created after finalizing a document or not. The RDY (Ready) file is created as the last file of the whole finalize process and can be used as a trigger for a follow-on process. Default: "true"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createSignatureImageDPI** (string): With this setting the resolution in DPI of the signature image which is save upon finalize of the document can be set. This is only applicable to image formats that contain DPI information e.g. TIFF, PNG,... The default resolution for HTML5 signatures is 96DPI and is reduced in relation to the default of 300. For example if the DPI is set to 100 the HTML5 signature resolution is also reduced to one third i.e. 32DPI. Default: "300"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createSignatureImageFile** (string): With this setting it is possible to decide, whether an image of the signature should be created after finalizing a document or not. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createSignatureImageFormat** (string): With this setting it is possible to define the format of the signature image which will be saved if sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createSignatureImageFile is set to true. Possible formats are "png", "gif", "bmp", "jpg", "tiff" (only supported for handwritten signatures and not for photos or Click-to-Sign signatures).Default: "png"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createTIFFCopy** (string): With this setting it is possible to decide, whether a TIFF copy of the signed document should be created. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createXMLFile** (string): With this setting it is possible to decide, whether an XML file should be created after finalizing a document. Default: "false"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.dmsFolder** (string): With this setting it is possible to change the path and the name of the folder where the finalized documents are stored on the SFTP Server. Default: "dms/de.softpro.sdweb.plugins.impl.SftpDms
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.fileDecoration** (string): The value of this setting is placed in all generated filenames after the docid (also after the optional timestamp) and before the optional filepostfix. A fileDecoration value could be useful for example if several SignDoc Web servers write their files to one common (shared) directory. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.filenamePostfix** (string): With this setting it is possible to define a postfix for all the files related to the document. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.filenamePrefix** (string): With this setting it is possible to define a prefix for all the files related to the document. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.password** (string): This is the password for the SFTP server used with the plugin. Default: ""
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.serverAddress** (string): This is the address for the SFTP server used with the plugin.Default: "localhost"

- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.serverPort** (integer): This is the port for the SFTP server used with the plugin. Default: 22
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.storeBiometricDataInXML** (string): With this setting it is possible to define whether the biometric signature data should be stored in the XML file or not. Only applicable if sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.createXMLFile="true". Default: "true"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.storeStrategy** (string): With this setting it is possible to decide, if the documents in the DMS should be stored in folders or not. If the storage strategy should be just files the value has to be set to "flat". Default: "folder"
- **sdwebplugins.de.softpro.sdweb.plugins.impl.SftpDms.user** (string): This is the user name for the SFTP server used with the plugin. Default: ""
- **supportedApps.blackList** (string list): With this parameter a blacklist can be defined, to block specific apps or devices. For further information see Related documentation: *SignDoc Web Administrator's Guide*, chapter "Check app environment". Default: []
- **supportedApps.whiteList** (string list): With this parameter a whitelist can be defined, to only allow specific apps or devices. For further information see Related documentation: *SignDoc Web Administrator's Guide*, chapter "Check app environment". Default: []

Chapter 7

Frequently asked questions

There are two cases, in which due to wrong configuration of the SignDoc Web server or the environment, the software does not work as expected. Both cases have one thing in common: The logfile is full of unusable stacktraces, since the problem is caused by misconfiguration and not by a server bug.

The messages have the goal, to put the administrator or user in the role to fix the problem on his own. The messages are displayed in the users language setting and in English.

Case 1 - session cookie gets lost

Usual suspect: Application server administrator / IT policies

Displayed message: The current cookie settings of the server (or in the browser) are possibly not correct. Cookies must be allowed for the web-context: /sdweb (http://localhost:8080/sdweb/). Another possible problem could be a load-balancer setup that is not configured for session affinity. Usually JSESSIONID is the session cookie that must be evaluated by a load-balancer.

Case 2 - esu parameter is used in a wrong way

Usual suspect: Website integrator / SignDoc Web administrator

Displayed message: The esu parameter was used in a wrong way! The URL of the HTTP request must match the selected URL that is defined by the esu parameter. Desired context defined by esu parameter: http://littlejoe:8080/sdweb/ - real request context : scheme=http serveNname=localhost serverPort=8080, contextPah=/sdweb

Other issues

- Error message: Cannot load SPFreeImage_1.dll, LoadLibrary failed, error 14001
  Probably the required MS VC++ runtime is not installed. One of the following runtimes is required.
  Microsoft Visual C++ Redistributable Runtime 32-bit:
  `http://www.microsoft.com/en-us/download/details.aspx?id=5582`
  Microsoft Visual C++ Redistributable Runtime 64-bit:
  `http://www.microsoft.com/en-us/download/details.aspx?id=2092`
  In doubt, install both C++ runtimes.
- Some (probably broken) fonts can crash the Sun/Oracle JVM (fontmanager.dll)
  This issue happens only on some systems if a broken font is used and the signature dialog is displayed.
  The effect is, that the fontmanager.dll of the Java Runtime crashes and terminates the server process.
  The issue occurs only on some CPU types, only with a i7 CPU and a JRE 1.6.
  This problem was so far not seen with a JRE 1.7.

"Keep Session Alive" mechanism

To keep a session on the server side alive, periodic HTTP GET requests can be executed on this URL:

```
<sdweb.external_server_url>/keepalive
```

Example

```
http://localhost:8080/sdweb/keepalive
```

Doing this will result in a HTTP status code as defined by setting sdweb.servlet.ignorelist.response_code (Default: 200 i.e. OK) and keep the session, referenced by the JSESSIONID header attribute, alive.