

Tungsten SignDoc Standard Administrator's Guide

Version: 3.4.0

Date: 2024-11-15

TUNGSTEN
AUTOMATION

© 2014–2024 Tungsten Automation. All rights reserved.

Tungsten and Tungsten Automation are trademarks of Tungsten Automation Corporation, 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 Tungsten Automation.

Table of Contents

Preface	6
Related documentation.....	6
Training.....	7
Getting help with Tungsten Automation products.....	7
Definitions.....	7
Chapter 1: Licensing	8
License handling.....	8
SignDoc license.....	8
Account license.....	9
Global license.....	9
Reset license (special license).....	10
Chapter 2: Logging	11
Chapter 3: Configuration	12
Configuration files.....	12
Common configuration options.....	12
Configure the 'autoprepate' functionality.....	13
Fonts used in the final document.....	13
Configuration service.....	14
Configuration levels.....	14
Change configuration options.....	15
Configuration values.....	18
Security.....	18
Email.....	19
Client.....	28
Advanced signing settings.....	34
AI (experimental).....	36
System.....	37
Documents and packages.....	47
Plug-ins.....	52
Webhooks.....	52
Chapter 4: Plug-ins	55
Plug-in handling.....	55
Plug-in administration.....	57
Plug-in development.....	62
Plug-in interface.....	62

Plug-in implementation.....	63
How SignDoc uses plug-ins.....	63
Signing plug-in.....	64
SigningEvent plug-in description.....	64
Minimal SigningEvent implementation.....	64
SigningRSA interface.....	64
Minimal SigningRSA implementation.....	65
Core plug-ins.....	65
Notification plug-in.....	70
Notification plug-in description.....	70
Core plug-ins.....	71
Package state change plug-in.....	76
Package state change plug-in description.....	76
Core plug-ins.....	78
Signer search plug-in.....	81
Signer search plug-in description.....	81
Core plug-ins.....	83
Document scan plug-in description.....	83
Core plug-ins.....	84
TSP plug-in.....	86
Trusted service provider plug-in description.....	86
TSP features with SignDoc.....	90
Chapter 5: Email.....	92
SMTP configuration.....	92
S/MIME configuration.....	93
Chapter 6: BankID.....	94
BankID Windows service configuration.....	94
Installing the external SignDocBankIdService.....	95
Configuration parameters.....	95
BankID authentication.....	96
BankID signing.....	97
Audit.....	99
Localization.....	99
References.....	99
Chapter 7: Tenant-specific URL.....	100
Preselect SignDoc account.....	100
Associate DNS label.....	100
Chapter 8: Generative AI.....	102

AI functionality..... 102

AI provider configuration..... 102

AI Tasks..... 103

Appendix A: External authentication using Microsoft and Google..... 104

Appendix B: Google Chrome Group Policy (GPO)..... 106

Appendix C: Font substitution when converting from MS Word to PDF..... 107

Preface

This guide provides information on SignDoc Standard licensing, logging, configuration, and plugins.

Related documentation

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

<https://docshield.tungstenautomation.com/Portal/Products/SD/3.4.0-rq5j77n6cd/SD.htm>

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

Release notes

- *Tungsten SignDoc Release Notes*

Technical specifications

- *Tungsten SignDoc Technical Specifications*

Guides

- *Tungsten SignDoc Standard Developer's Guide*
- *Tungsten SignDoc Standard Installation Guide*

Help

- *Tungsten SignDoc Standard Help*
- *Tungsten SignDoc Standard Administration Center Help*
- *Tungsten SignDoc Assistant Help*
- *Tungsten SignDoc Help - Signing Documents*
- *Tungsten SignDoc Device Connector Help*
- *Tungsten SignAlyze Help*

Software development kit

- *Tungsten SignDoc Browser Capture Help*
- *Tungsten SignDoc SDK API Documentation (C)*
- *Tungsten SignDoc SDK API Documentation (C++)*
- *Tungsten SignDoc SDK API Documentation (.NET with exceptions)*
- *Tungsten SignDoc SDK API Documentation (.NET without exceptions)*
- *Tungsten SignDoc SDK API Documentation (Java)*


Training

Tungsten Automation offers both on-demand and instructor-led training to help you make the most of your product. To learn more about training courses and schedules, visit the [Tungsten Automation Learning Cloud](#).

Getting help with Tungsten Automation products

The [Tungsten Automation Knowledge Portal](#) repository contains articles that are updated on a regular basis to keep you informed about Tungsten Automation products. We encourage you to use the Knowledge Portal to obtain answers to your product questions.

To access the Tungsten Automation Knowledge Portal, go to <https://knowledge.tungstenautomation.com/>.

 The Tungsten Automation Knowledge Portal is optimized for use with Google Chrome, Mozilla Firefox, or Microsoft Edge.

The Tungsten Automation Knowledge Portal 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.
To locate articles, go to the Knowledge Portal home page and select the applicable Solution Family for your product, or click the View All Products button.

From the Knowledge Portal home page, you can:

- Access the Tungsten Automation Community (for all customers).
On the Resources menu, click the **Community** link.
- Access the Tungsten Automation Customer Portal (for eligible customers).
Go to the [Support Portal Information](#) page and click **Log in to the Customer Portal**.
- Access the Tungsten Automation Partner Portal (for eligible partners).
Go to the [Support Portal Information](#) page and click **Log in to the Partner Portal**.
- Access Tungsten Automation support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.
Go to the [Support Details](#) page and select the appropriate article.

Definitions

- <INSTALLDIR> is the directory with the unpacked signdoc-standard-3.4.0.zip file. See "Quick start procedure" in the *Tungsten SignDoc Standard Installation Guide*.
- <CIRRUS_HOME> is the directory of the SignDoc Standard web application.

Chapter 1

Licensing

This chapter contains information on the SignDoc license. It describes how to request a SignDoc licence and how to handle the different license types.

License handling

To use SignDoc Standard it is required to install a valid SignDoc license.

For on-premise installations the SignDoc license file `KofaxSignDoc.key` will be provided by the Tungsten Automation Order Fulfillment team based on a sales order. You will get a set of license files which is specially made out for the customer.

SignDoc license

To run SignDoc an appropriate license key file has to be installed. The license key contains information about the permissions. The key file can be opened in a text editor. The order of the license defines the permissions.

A SignDoc license can be installed only if the following requirements are met:

- It must be a valid SignDoc license
- The license is not expired

A SignDoc license `KofaxSignDoc.key` can either be used for an individual account or for all accounts of an installation.

Example for the content of a SignDoc license key:

```
h:SPLM2 4.10
i:ID:9923379
i:Product:TungstenSignDoc
i:Manufacturer:Tungsten Automation Deutschland GmbH
i:Customer:Dummy corporation
i:Version:99
i:OS:all
a:product:unlimited
a:signware:unlimited
a:sign:2024-12-31
a:capture:unlimited
a:render:unlimited
ps:ACCOUNT_INFORMATION:F4D22WO65F
ps:CustomerCompany:Dummy corporation
ps:COUNTER_RENEWAL_PERIOD:MONTHLY
```



```

pi:LICENSE_TYPE:1
pi:MAX_SIGNING_PACKAGES:100000
pi:MAX_USERS:100
s:7372e410285f21fcd7cecd5669190394f951fa032889e97be8f7bdad2c8091ef
s:7c414338328c13d0ace0e55501fa640e97ed322b5f89a941355212223f4e8c84
s:3914e3749ffd5e80c49c592ef9f89819f11774b84355d285ca0f50c8d54d732e
s:79ae6555d940d0c7f06bfec65714e33985db5f306e897f6e05e91516c546b25d

```

Explanation of the application-related license parameters:

- `a:sign:2024-12-31` contains the date of expiry.
- `pi:MAX_SIGNING_PACKAGES:100000` describes the number of licensed signing packages.
- `pi:MAX_USERS:100` shows the maximum number of SignDoc users that can be created.
- `ps:ACCOUNT_INFORMATION:F4D22W065F` is a customer-unique string and is important if you need an upgrade of the license, for example if you need more packages or if you want to increase the number of users. Only a license with the same `ps:ACCOUNT_INFORMATION` entry can be used for a subsequent license import for the account. Once installed, this `ps:ACCOUNT_INFORMATION` is dedicated for a specific account, that means, that a license with the same `ps:ACCOUNT_INFORMATION` cannot be used for more than one account.
- `ps:COUNTER_RENEWAL_PERIOD:MONTHLY` causes a monthly package counter reset (default is YEARLY).

Account license

A SignDoc license will become an account license when the license file is applied for an individual account of the SignDoc installation. The account license includes permissions which are only valid for this specific SignDoc account. Any other accounts in the system need a separate license in each case. An account license must be installed during account creation. It is not possible to install an account license separately from the account creation. It is only possible to upgrade the license afterwards.

An account license can be installed for the first time only by a SignDoc server administrator during account creation. An upgrade of an account license can be installed by a SignDoc server administrator or by an account administrator.

An account license can be upgraded only with another license with the same `ps:ACCOUNT_INFORMATION` entry. Additionally, the following is required:


- It must be a valid SignDoc license.
- The license is not expired.
- The number of already processed packages must not exceed the number of licensed packages.
- The number of already existing account-specific users must not exceed the number of licensed users.

Global license

A SignDoc license will become a global license when the license file is not applied for a specific account. In this case, the license can be used as a system-wide license which is valid for all accounts.

The permissions included are valid for all accounts without the necessity of an individual license for each account. The licensed number of packages and/or users are shared between all accounts.

A global license can be installed by a SignDoc server administrator before any account is created. If a global license is installed an arbitrary number of accounts can be created. No additional account-specific licenses are needed.

 Once a global license is installed it is not possible to return back to account-specific licenses! In general, if a global license is installed, any installed account-specific licenses are invalidated or rather removed.

A global license can be upgraded only with another license with the same `ps:ACCOUNT_INFORMATION` entry. Additionally, the following is required:

- It must be a valid SignDoc license.
- The license is not expired.
- The number of already processed packages must not exceed the number of licensed packages.
- The number of already existing account specific users must not exceed the number of licensed users.

Use an installed account license as global license

It is possible to use an applied account license as a global license for the system. When the license key file will be applied again as global license the number of already processed packages for this account is used as base number of packages for the global license. All subsequent created signing packages (including templates) are counted on a base of this number. During installation of the global license, it is checked whether the total number of all account-related users exceeds the number of licensed users. In a case of excess the license cannot be installed.

Apply a global license as replacement for already installed account licenses

It is possible to install a new SignDoc license as a global license even if you have installed any account-specific licenses.

Reset license (special license)

A reset license is a special license which must be requested from Tungsten Automation if the installed license is invalid or corrupted for any reason. Since this is an unrecoverable problem, it requires special treatment. If an account license or a global license is corrupted, it is required to 'reset' the license by this specific reset license. Only the import of a reset license allows the application to continue with the installation of a valid 'normal' license (account or global license).

It is not necessary (and also not possible) to use a reset license if the already installed license is valid or expired or if any maximum (of users or packages) is reached. A reset license is a time restricted license file which cannot be used as a production license.

Chapter 2

Logging

SignDoc Standard is able to trace any application messages into a log. This log is independent from the audit trail which records package processing steps into the SignDoc Standard database.

Logging is configured in the Administration Center by these settings

- **signdoc.logger.level**

Sets the base log level for all SignDoc specific components. A level more verbose than INFO (i.e., DEBUG | FINE | FINER | FINEST | TRACE | ALL) might impact the performance of the system. Valid log levels: OFF | FATAL | SEVERE | ERROR | WARNING | WARN | INFO | CONFIG | DEBUG | FINE | FINER | FINEST | TRACE | ALL

Default value: INFO

- **signdoc.logger.other_levels**

Sets the base log level for all non SignDoc components like e.g. Spring and Tomcat. A level more verbose than INFO (i.e., DEBUG | FINE | FINER | FINEST | TRACE | ALL) might impact the performance of the system. Valid log levels: OFF | FATAL | SEVERE | ERROR | WARNING | WARN | INFO | CONFIG | DEBUG | FINE | FINER | FINEST | TRACE | ALL

Default value: INFO

- **signdoc.logger.custom_levels**

Provides customized log levels that overwrite the default log level defined by `signdoc.logger.level`. Use one line per logger. A leading # comments the line. Valid log levels: OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL. Syntax: `LOGGER_NAME=LOG_LEVEL`

Default value:

#REST API exceptions

#de.softpro.cirrus.rest.common.controller.BaseRestController=FINE


#REST API requests

#de.softpro.cirrus.web.helper.RequestLogFilter=FINER

#Apache Tomcat logs

#org.apache.tomcat=INFO

#org.apache.catalina=INFO

 It's strongly discouraged to set log level FINE, FINER, FINEST or ALL for the whole server since it might impact overall performance. It's recommended to make use of `signdoc.logger.custom_levels` if a more detailed log than INFO is needed.

Logging features such as log file rotation and or limiting the log file size can be configured and adjusted in file `logback-spring.xml` by using standard features of the logback logging system.

Chapter 3

Configuration

This chapter describes the files, services and values used for configuring SignDoc.

Almost all configuration is done using Configuration Service of the Administration Center or SignDoc Account, but at least the database connection must be configured either by Environment Variables or modifying the `application.properties` file. This is described in more detail in the *SignDoc Installation Guide*.

Configuration files

The following properties are not essential, but can still be configured by editing the `application.properties` file which replaces the `service_configuration.properties` file used in previous SignDoc Standard version. Note that these properties are optional and may be absent in the current `application.properties` file.

Common configuration options

- **cirrus.rest.hide.app.version**

If set to true, the REST API will not reveal the application version information without authentication. Supported values: true, false. Default: false

Example

```
cirrus.rest.hide.app.version=false
```

For information on LDAP properties, see "Authentication LDAP" in the *Tungsten SignDoc Standard Installation Guide*.

The number of REST requests for a user in a time frame can be restricted using the configuration options below. Both configurations should have a positive value for the rate limiting to work.

- **cirrus.rest.request.max.size**

Defines the number of REST requests allowed for a user with an authentication token. Default value: -1, allows infinite requests.

Example

```
cirrus.rest.request.max.size=-1
```

- **cirrus.rest.request.max.size.time**


Time frame in seconds to restrict the maximum number of REST requests for the configuration `cirrus.rest.request.max.size`. Default value: -1, allows infinite requests.

Example

```
cirrus.rest.request.max.size.time=-1
```

- **cirrus.signdoc.assistant.domain**

The SignDoc Assistant domain along with API version.

 This is an internal setting that should not be changed without advice from Technical Support.

Example

```
https://{host}/api/v1/
```

Configure the 'autoprepate' functionality

During preparation of a signing package the user may choose to 'autoprepate' a document after the upload. This means that for every defined signer a signature field is added to the document automatically.

Signature fields are placed in rows and columns starting at the top/left margin. If a field exceeds the right margin, it will be placed in the next row. If there is no space left on the page, an exception is thrown.

Properties used for configuring the 'autoprepate' functionality

Add configuration properties to `INSTALLDIR\application.properties` configuration file.

- **cirrus.autoprepate.option**
Autoprepate first or last page, defaults to first.
- **cirrus.autoprepate.margin.top**
The top margin of the page where signature fields are placed.
- **cirrus.autoprepate.margin.left**
The left margin of the page where signature fields are placed.
- **cirrus.autoprepate.margin.bottom**
The bottom margin of the page where signature fields are placed.
- **cirrus.autoprepate.margin.right**
The right margin of the page where signature fields are placed.
- **cirrus.autoprepate.padding.horiz**
The space between rows of signature fields.
- **cirrus.autoprepate.padding.vert**
The space between columns of signature fields.

All numeric values are in pixel units. The default value is always 10.

Fonts used in the final document

When building the final document, the application searches for compatible fonts in `INSTALLDIR\signdoc_home\fonts` directory by default. Copy any fonts that might be used in the application in that location.

The font for the final package PDF document's can be controlled by the system administrator, using pre-defined configurations for font name and font directory.

- **cirrus.document.signing.final-package.font-directory**

Defines the location where fonts are available, the path should be a valid path on the server and should contain the font defined in the font name. If this path is not explicitly defined by the system administrator, then `INSTALLDIR\signdoc_home\fonts` is used.

- **cirrus.document.signing.final-package.font-name**

Defines the font name that should be used for the final document. The user has flexibility to choose his/her own font. The font name defined here should be either available in the font directory or in the system. By default, 'Noto Sans' is used.

When the final document is created, if the font is no longer valid (such as if the font folder is deleted) then the application falls back to the default font and font directory. If there are issues locating the default font and font directory the final package is not assembled and the email containing the final document will not be sent.

Configuration service

The reasons for using a configuration service are:

- Changing configuration options on the fly, without a server restart.
- Setting configuration values individually on an account basis.
- Providing help and validation on configuration values via a GUI.

Configuration levels

SignDoc Standard has the following configuration levels:

- **Default value**
The default value for a configuration option can be provided by the application. This value applies if no configuration is set by the user. This value cannot be changed.
- **Global (application-wide) value**
This value can be set by the server administrator and applies to all accounts.
- **Account-specific value**
This value can be set by an account administrator (permission dependent) or by the server administrator. It only applies to the account it has been set for. Not all configuration values can be set account specific.

The application uses the following precedence when determining the value to apply for one configuration option and account:

1. If an account-specific value exists, it is used.
2. If no account-specific value exists, the global value is used if available.
3. If no global value exists, the default value is used.
4. If no default value is defined, no value is returned.

Change configuration options

All configuration service values are set using the REST API. Both the Administration Center and the Manage Client provide user interfaces to view and edit configuration values. They use the underlying REST API to apply the changes.

Configuration using the Administration Center

Server administrators use the Administration Center to change configuration values. They can change both global and account-specific values.

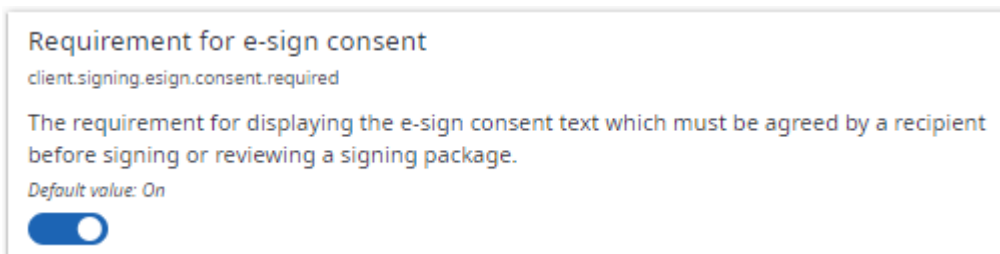
From the Administration Center starting page, click the **System settings** link in the navigation panel to change global values.

The **System settings** menu is displayed which includes settings related to the system, documents and packages, plug-ins, security, email, client and advanced signing settings.

The configuration options are grouped in categories shown on the left.

Each configuration option will have:

- A title
- The configuration option ID
- A description of the configuration option
- A default value if provided



The entry field is used to edit the configuration option value. The configuration value on the current level (in this case global) is shown in normal text. If it is not set and a lower precedence level (in this case default) exists, that value will be shown grayed out. Also, the default value if provided is always displayed above the entry field.

E-sign consent text

client.signing.esign.consent.text

The e-sign consent text which must be agreed by a recipient before signing or reviewing a signing package.

Show default value ▾

If you use this application, your signature (in a written form or otherwise if relevant for the signing process) will be processed electronically. By selecting the "I agree to use electronic signature" acknowledgment you confirm that you agree to sign documents electronically, and that you understand that the provision of such

A configuration value will be validated by the client before it is used.

Multiple configuration values can and should be set in one go. Related configuration values should be changed together. When everything is set, clicking **Save** will store the new configuration on the server.

Account-specific configuration options can be set by first selecting the account:

Manage accounts

[Create account](#)

	ID ^	License state ↑	# Users ↑	# Packages ↑	State ↑	Options
<input type="checkbox"/>	basic	Valid	20	0	Active	Edit

Clicking **Edit** will take you to the page where the account-specific settings can be edited.

Configuration using the Manage Client

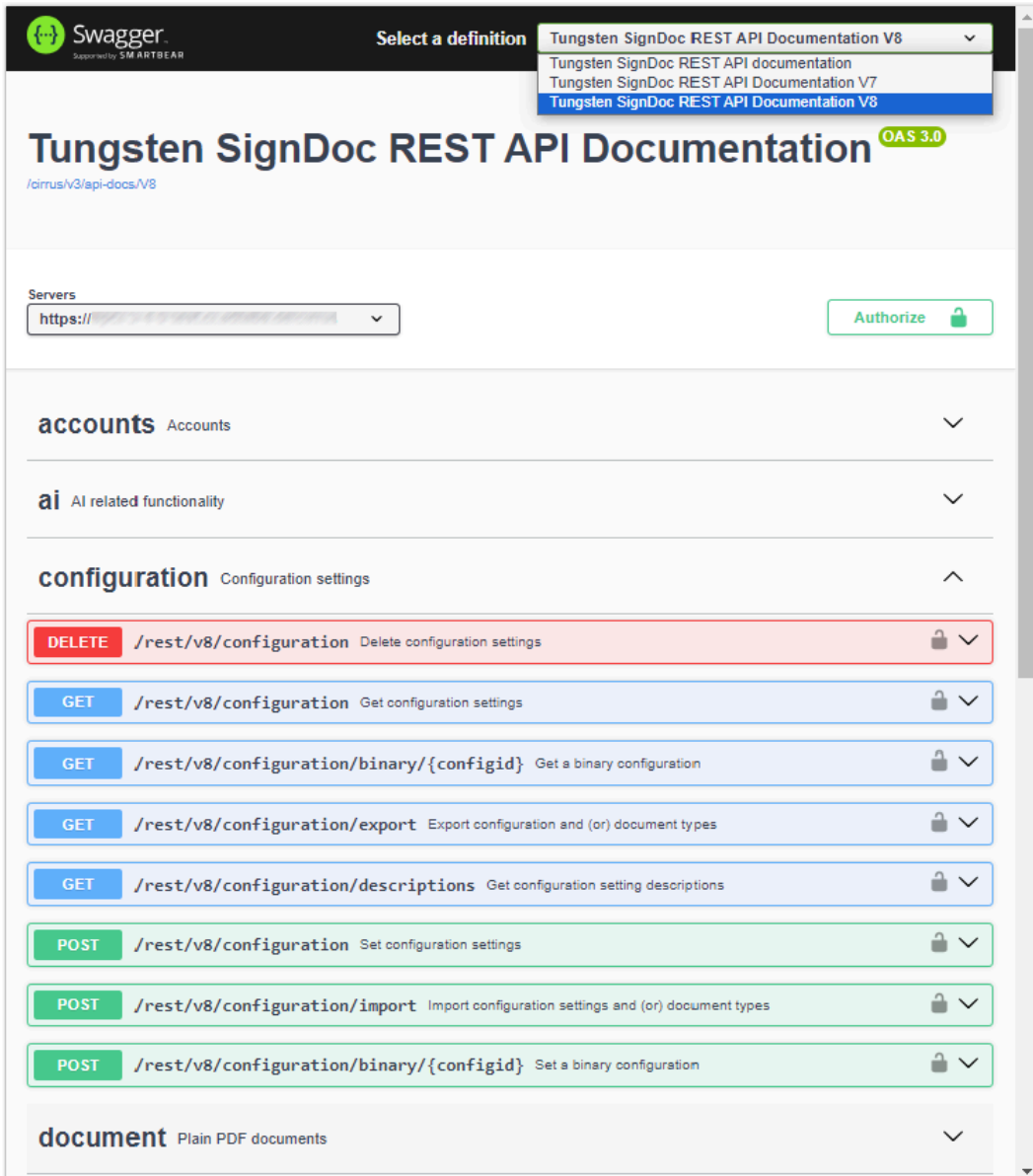
Account administrators will use the Manage Client to change configuration settings.

From the Manage Client starting page, click **Administration** in the title bar to get to the configuration settings.

The configuration options are grouped in categories shown on the left. The display and editing of individual configuration options is similar to the Administration Center.

Configuration using the REST API

All configuration options can also be set programmatically using the REST API.



The screenshot displays the Swagger UI for the Tungsten SignDoc REST API. The top navigation bar includes the Swagger logo and a dropdown menu for selecting a definition, currently set to 'Tungsten SignDoc REST API Documentation V8'. The main heading is 'Tungsten SignDoc REST API Documentation' with a version indicator 'OAS 3.0'. Below the heading, there is a 'Servers' section with a dropdown menu showing 'https://' and an 'Authorize' button. The main content area is divided into sections: 'accounts', 'ai', 'configuration', and 'document'. The 'configuration' section is expanded, showing a list of REST API endpoints. The 'DELETE /rest/v8/configuration' endpoint is highlighted in red, indicating it is the current selection. Other endpoints include GET, POST, and GET methods for various configuration settings.

Method	Endpoint	Description	Auth	Expand
DELETE	/rest/v8/configuration	Delete configuration settings	🔒	⌵
GET	/rest/v8/configuration	Get configuration settings	🔒	⌵
GET	/rest/v8/configuration/binary/{configid}	Get a binary configuration	🔒	⌵
GET	/rest/v8/configuration/export	Export configuration and (or) document types	🔒	⌵
GET	/rest/v8/configuration/descriptions	Get configuration setting descriptions	🔒	⌵
POST	/rest/v8/configuration	Set configuration settings	🔒	⌵
POST	/rest/v8/configuration/import	Import configuration settings and (or) document types	🔒	⌵
POST	/rest/v8/configuration/binary/{configid}	Set a binary configuration	🔒	⌵

Using the REST API is documented in the *Tungsten SignDoc Standard Developer's Guide*.

Configuration values

The following configuration settings can be accessed in the Manage Client and/or in the Administration Center.

Security

This section lists the security related settings.

- **Signer blocked time**

`security.fail.accesscode.blocked.time`

Period of time in milliseconds a signer user is blocked before next 2-factor authentication attempt is allowed. This is effective for signers that were blocked after unsuccessful authentication attempts.

Default value: 360000

- **Maximum number of failed signer authentication attempts**

`security.fail.accesscode.max.attempts`

Maximum number of failed signer 2-factor authentication attempts allowed until the signer is blocked. See `security.fail.accesscode.blocked.time`.

Default value: 5

- **Reset duration after unsuccessful signer access code authentication**

`security.fail.accesscode.max.life.seconds`

Maximum number of seconds before the counter for unsuccessful 2-factor authentication attempts for a signer is reset (as long as the signer is not blocked). The max life time is counted after first failed authentication with a wrong access code. If further authentication attempts fail within the max life time, the signer is blocked. See `security.fail.accesscode.blocked.time`.

Default value: 86400

- **Action after too many failed user authentication trials**

`security.fail.login.action`

Action to be taken if maximum number of failed user authentications is reached, supported values are SUSPEND and BLOCK. SUSPEND will suspend the user, only an administrator can activate the user again. BLOCK will block any authentication attempts. The period of blocked time is set by configuration parameter `security.fail.login.blocked.time`. A reactivation by an administrator is not possible during this time period. This is also effective for users that could not be suspended. A user cannot be suspended if he is the last user with role ADMIN for an account or if the affected user is the last SUPERUSER in the system.

Default value: SUSPEND

- **User blocked time**

`security.fail.login.blocked.time`

Period of time in milliseconds a user is blocked before next authentication attempt is allowed. This is effective only for users that were blocked after unsuccessful login attempts.

Default value: 360000

- **Maximum number of failed login attempts**

`security.fail.login.max.attempts`

Maximum number of failed login attempts allowed until a user is suspended or blocked. See `security.fail.login.action`.

Default value: 10

- **Reset duration after unsuccessful user authentication**

`security.fail.login.max.life.seconds`

Maximum number of seconds before the counter for unsuccessful attempts for a user is reset (as long as the user is not suspended or blocked). The max life time is counted after first failed authentication. If further authentication attempts fail within the max life time, the user is suspended or blocked. See `security.fail.login.action`.

Default value: 86400

- **Add additional HTTP response headers**

`security.http.response.headers.add`

Add additional HTTP Headers. Each line must contain a header entry in the format `<HEADER_NAME>:<HEADER_VALUE>`. Example: `MY_HEADER:MY_VALUE`

- **Set HTTP response headers**

`security.http.response.headers.set`

Set (and overwrite if necessary) existing HTTP Headers. Each line must contain a header entry in the format `<HEADER_NAME>:<HEADER_VALUE>`. Example: `MY_HEADER:MY_VALUE`

Email

This section lists the email related settings.

S/MIME related settings

- **S/MIME Certificate**

`mail.s-mime.certificate`

Certificate in PKCS12 format which will be used to sign the MIME data.

General email settings

- **Default communication language**

`cirrus.communication.locale`

Default communication language which is used for email notifications. It must be a valid IETF BCP 47 language tag. See <https://www.rfc-editor.org/info/bcp47>. Example: `en` for English or `pt-BR` for Brazilian Portuguese.

Default value: `en`

- **Delegate link in email**

`cirrus.mail.delegate.enabled`

If set to 'On', the delegate link is visible to the recipient in the email, else not shown.

Default value: `On`

- **Default delegate message**

`cirrus.mail.delegate.message.default`

The default message which is sent when a signer delegates the signing session to another person. The placeholders `$DELEGATED_SIGNER` and `$SIGNER` can be used. `$DELEGATED_SIGNER`

is replaced by the name of the person to which the signing session is delegated. \$SIGNER is replaced by the name of the current signer.

Default value: Hi \$DELEGATED_SIGNER, I am forwarding this signing session to you as it does not fall within my remit. If you are not the correct person, please use the "Delegate" link in this email to forward it either to someone else or back to me again. Regards, \$SIGNER.

- **Delegate message prefix**

`cirrus.mail.delegate.message.prefix`

This message is prefixed to the actual delegate message provided by the recipient when delegating the signing session.

Default value: Message from the original recipient

- **Final document notification**

`cirrus.mail.finaldocument.notification.enabled`

Email notifications for the final document.

Default value: On

Currently a Final Document is sent automatically to all signers with an email address after completion of a package.

The Final Document contains all signed documents and audit trails of a package.

The default of this account-specific setting

`cirrus.mail.finaldocument.notification.enabled=[true|false]` is true.


The roles ADMIN and SUPERUSER have read/write access to this setting.

If the setting is set to false, no email notification is sent to any of the included recipients if a package is completed.

There are 2 locations for the setting that control the behavior:

[Configuration Database] `cirrus.mail.finaldocument.notification.enabled=[true|false]`

[application.properties file] `cirrus.mail.finaldocument.notification.enabled=[true|false]`

 A configuration setting in the configuration database takes precedence over a setting in application.properties file.

- **Email debug**

`mail.debug`

Sets debug for email. Setting the value to 'On' will cause JavaMail to print debugging messages as it attempts to load each configuration file.

Default value: Off

- **Send an email if a user password has been changed**

`mail.enabled.password.changed`

Send a notification email to the user if his password has been changed.

Default value: Off

- **Access code text for missing delivery plug-in**

`mail.message.accesscode.error`

The access code text for a missing or removed delivery plug-in. No placeholder expansion is performed here, since this is a placeholder content.

Default value: Access code required to open the document(s) cannot be delivered automatically. Please contact the signing package owner.

- **Access code text for manual delivery**

`mail.message.accesscode.manual`

The access code text for a manual access code delivery. No placeholder expansion is performed here, since this is a placeholder content.

Default value: You will need to enter an access code to open the document(s). The signing package owner will provide it.

- **Access code text for delivery by plug-in**

`mail.message.accesscode.plugin`

The access code text for delivery by plug-in. The only placeholder allowed is `%NOTIFICATIONPLUGINTYPE%` which will query the plug-in type used.

Default value: You will need to enter an access code to open the document(s). It will be sent to you by `%%NOTIFICATIONPLUGINTYPE%%` when you start the signing session.

- **Account disabled email body**

`mail.message.account.disabled.body`

The body for account disabled emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Account disabled email subject**

`mail.message.account.disabled.subject`

The subject line for account disabled emails.

Default value: Your login to Tungsten SignDoc was disabled for security reasons

- **User invitation email body**

`mail.message.account.invited.body`

The body for user invitation emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **User invitation email subject**

`mail.message.account.invited.subject`

The subject line for user invitation emails.

Default value: You have been invited to join Tungsten SignDoc

- **Changed password email body**

`mail.message.changed.password.body`

The body for changed password emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Changed password email subject**

`mail.message.changed.password.subject`

The subject line for changed password emails.

Default value: Tungsten SignDoc password changed

- **Document copy email body**

`mail.message.email.me.a.copy.body`

The body for document copy emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Document copy email subject**

`mail.message.email.me.a.copy.subject`

The subject line for document copy emails.

Default value: Your documents

- **Password forgotten email body**

`mail.message.forgotten.password.body`

The body for password forgotten emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Password forgotten email subject**

`mail.message.forgotten.password.subject`

The subject line for password forgotten emails.

Default value: Tungsten SignDoc password reset

- **Recipient delegate email body**

`mail.message.inform.owner.about.delegation.body`

The body of the email to the owner of the signing package sent when a recipient delegates the signing session.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Recipient delegate email subject**

`mail.message.inform.owner.about.delegation.subject`

The subject line of the email to the owner of the signing package sent when a recipient delegates the signing session.

Default value: Recipient has delegated the signing package.

- **Reviewer complete email body**

`mail.message.inform.owner.aboutReviewer.complete.body`

The body for owner email after reviewer complete.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Reviewer complete email subject**

`mail.message.inform.owner.aboutReviewer.complete.subject`

The subject line for owner email after reviewer complete.

Default value: Reviewer has completed the signing package

- **Signer complete email body**

`mail.message.inform.owner.aboutSigner.complete.body`

The body for owner email after signer complete.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Signer complete email subject**

`mail.message.inform.owner.about.signer.complete.subject`

The subject line for owner email after signer complete.

Default value: Signer has completed the signing package

- **Manager added to a team email body**

`mail.message.manager.add.to.team.body`

The body for an email when a manager is successfully added to a team.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Manager added to a team email subject**

`mail.message.manager.add.to.team.subject`

The subject line for an email when a manager is successfully added to a team.

Default value: Tungsten SignDoc - You are added as team manager to a team

- **Package complete owner email body**

`mail.message.package.complete.owner.body`

The body for owner email after package complete.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Package complete owner email subject**

`mail.message.package.complete.owner.subject`

The subject line for owner email after package complete.

Default value: Signing Package "%%PACKAGENAME%%" completed

- **Package complete recipient email body**

`mail.message.package.complete.recipient.body`

The body for recipient email after package complete.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Package complete recipient email body for document download**

`mail.message.package.complete.recipient.download.body`

The body for recipient email after signing package complete with download documents option.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Package complete recipient email subject**

`mail.message.package.complete.recipient.subject`

The subject line for recipients email after package complete.

Default value: Signing Package "%%PACKAGENAME%%" completed

- **Decline reason text R1 (documents problem)**

`mail.message.reason.R1`

The decline reason text for R1 (documents problem). No placeholder expansion is performed here, since this is a placeholder content.

Default value: There is a problem with the document(s)

- **Decline reason text R2 (sender not recognized)**

`mail.message.reason.R2`

The decline reason text for R2 (sender not recognized). No placeholder expansion is performed here, since this is a placeholder content.

Default value: I do not recognize the sender

- **Decline reason text R3 (no online signing)**

`mail.message.reason.R3`

The decline reason text for R3 (no online signing). No placeholder expansion is performed here, since this is a placeholder content.

Default value: I do not want to sign online

- **Decline reason text R4 (unacceptable terms)**

`mail.message.reason.R4`

The decline reason text for R4 (unacceptable terms). No placeholder expansion is performed here, since this is a placeholder content.

Default value: I do not agree with the terms of the e-sign consent

- **Decline reason text R5 (unacceptable terms of GDPR statement)**

`mail.message.reason.R5`

The decline reason text for R5 (unacceptable terms of GDPR statement). No placeholder expansion is performed here, since this is a placeholder content.

Default value: I do not agree with the terms of the GDPR statement

- **Signer declined email body**

`mail.message.rejected.body`

The body for signer declined emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Signer declined email subject**

`mail.message.rejected.subject`

The subject line for signer declined emails.

Default value: Signer has declined the signing of a package

- **Reminder email notification body**

`mail.message.reminder.body`

The body for reminder email notifications.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Reminder email notification subject**

`mail.message.reminder.subject`

The subject for reminder email notifications.

Default value: %%PACKAGEMAILSUBJECT%%

- **Password reset email body**

`mail.message.reset.password.body`

The body for password reset emails.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Password reset email subject**
`mail.message.reset.password.subject`
The subject line for password reset emails.
Default value: Tungsten SignDoc password reset
- **Reviewer notification email body**
`mail.message.reviewing.body`
The body for reviewer notification emails.
The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.
- **Reviewer notification email subject**
`mail.message.reviewing.subject`
The subject for reviewer notification emails.
Default value: %%PACKAGEMAILSUBJECT%%
- **Custom reminder email body**
`mail.message.send.message`
The body for custom reminder emails.
The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.
- **Custom reminder with link email body**
`mail.message.send.message.with.link`
The body for custom reminder with link emails.
The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.
- **Signer notification email body**
`mail.message.signing.body`
The body for signer notification emails.
The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.
- **Signer notification email subject**
`mail.message.signing.subject`
The subject for signer notification emails.
Default value: %%PACKAGEMAILSUBJECT%%
- **Default value for signer notification email subject**
`mail.message.signing.subject.default`
The default subject text for package specific signer notification emails. When necessary, this string can contain special characters that are replaced with meaningful data at runtime. Available special characters are \$USER which is replaced by the current user name and \$NOW which is replaced by the current time.
Default value: An e-sign Request via Tungsten SignDoc
- **Default value for signer notification email text**
`mail.message.signing.text.default`
The default value for the package specific message in signer notification emails. When necessary, this string can contain special characters that are replaced with meaningful data at runtime.

Available special characters are \$USER which is replaced by the current user name and \$NOW which is replaced by the current time.

Default value: \$USER has invited you to e-sign documents with Tungsten SignDoc.

- **User added to team email body**

`mail.message.user.add.to.team.invited.body`

The body for an email when a user is successfully added to a team.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **User added to team email subject**

`mail.message.user.add.to.team.invited.subject`

The subject line for an email when a user is successfully added to a team.

Default value: Tungsten SignDoc - Welcome to the Team!

- **Team invite accept note for user**

`mail.message.user.invite.to.team.acceptnote`

A note to the user with information of the consequences of joining a team.

Default value: Upon joining your SignDoc team, you will be able to read, edit, and send all of the signature packages available to the team. Your signing packages and documents will be shared with other team members.

- **Join team invitation email body**

`mail.message.user.invite.to.team.body`

The body for inviting a user to join a team.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **Join team invitation email subject**

`mail.message.user.invite.to.team.subject`

The subject line for inviting a user to join a team.

Default value: Tungsten SignDoc - You have been invited to join a team

- **User self remove from a team sent to all the team managers email body**

`mail.message.user.selfremove.from.team.body`

The email body reporting all the managers of a team when a user disassociates itself from a team.

The default value is the corresponding HTML text displayed in the available online HTML editor below the setting.

- **User self remove from team email subject**

`mail.message.user.selfremove.from.team.subject`

The subject line for the email sent to all the team managers when a user removes itself from a team

Default value: Tungsten SignDoc - User is removed from the team

SMTP related settings

- **Start up Email**

`cirrus.startup.email`

Defined destination recipient email address to receive email on startup. Requires valid functional SMTP system configuration, will not work if only account specific SMTP configurations are available.

- **Host**

`mail.smtp.host`

The domain name of the SMTP server.

- **Port**

`mail.smtp.port`

The port of the SMTP server.

- **User**

`mail.smtp.user`

User name to connect to SMTP server to use SMTP authentication.

- **Password**

`mail.smtp.password`

User password to connect to SMTP server to use SMTP authentication.

- **From**

`mail.smtp.from`

Default 'From' email address. Please use a valid email address.

- **StartTLS enabled**

`mail.smtp.starttls.enable`

Enables or disables SMTP StartTLS setting.

Default value: On

- **StartTLS required**

`mail.smtp.starttls.required`

Enables or disables SMTP StartTLS required setting.

Default value: On

- **SSL Check Server identity**

`mail.smtp.ssl.checkserveridentity`

Enables or disables SMTP check server identity setting.

Default value: On

- **SMTP Authentication**

`mail.smtp.auth`

Defines if SMTP connection should be authenticated. If the value is not provided in the setting, it is derived from user name and password. If user name and password are set, this is 'true' by default.

- **SMTP connection timeout**

`mail.smtp.connectiontimeout`

Socket connection timeout value in milliseconds for opening a SMTP socket connection. This timeout is implemented by `java.net.Socket`. Default is infinite timeout.

- **SMTP local host name**

`mail.smtp.localhost`

Local host name used in the SMTP HELO or EHLO command. Defaults to `InetAddress.getLocalHost().getHostName()`. Should not normally need to be set if your JDK and your name service are configured properly, however useful to avoid issues with problematic DNS settings.

- **SMTP connection I/O timeout**

`mail.smtp.timeout`

Socket I/O timeout value in milliseconds. This timeout is implemented by `java.net.Socket`. Default is infinite timeout.

- **SMTP connection write timeout**

`mail.smtp.writetimeout`

Socket write timeout value in milliseconds. This timeout is implemented by using a `java.util.concurrent.ScheduledExecutorService` per connection that schedules a thread to close the socket if the timeout expires. Thus, the overhead of using this timeout is one thread per connection. Default is infinite timeout.

Client

This section lists the client related settings.

Signing Client related settings

- **Display first page of document**

`client.signing.document.display.firstpage.enabled`

If set to 'On', the first page of a document is always displayed when opened in the Signing Client. If set to 'Off', the page which contains the first interactive field is displayed when a document is opened in the Signing Client.

Default value: Off

- **Requirement for e-sign consent**

`client.signing.esign.consent.required`

The requirement for displaying the e-sign consent text which must be agreed by a recipient before signing or reviewing a signing package.

Default value: On

- **E-sign consent text**

`client.signing.esign.consent.text`

The e-sign consent text which must be agreed by a recipient before signing or reviewing a signing package.

Default value: If you use this application, your signature (in a written form or otherwise if relevant for the signing process) will be processed electronically. By selecting the "I agree to use electronic signature" acknowledgment you confirm that you agree to sign documents electronically, and that you understand that the provision of such signatures constitutes a legally binding transaction according to the applicable local e-sign legislature.

- **The external e-sign consent URL**

`client.signing.esign.consent.url`

The custom e-sign consent URL which is provided as link in the Signing Client in addition to the e-sign consent text (max 2000 chars).

- **Requirement for GDPR consent**

`client.signing.gdpr.required`

The requirement for displaying the GDPR (General Data Protection Regulation) consent text which must be agreed by a recipient before signing or reviewing a signing package.

Default value: Off

- **GDPR statement**

`client.signing.gdpr.text`

GDPR (General Data Protection Regulation) text which must be agreed by a recipient before signing or reviewing a signing package.

Default value: If you use this application, your personal data will be processed. This may include biometric data if relevant for the signing process. For more information, please read the Privacy Notice and Consent carefully. By selecting the "I agree to the data protection policy" acknowledgment you confirm (1) that you have read the Privacy Notice and Consent, and (2) that you consent to the use of biometric data if relevant for the signing process.

- **The external GDPR policy URL**

`client.signing.gdpr.url`

The custom GDPR (General Data Protection Regulation) URL which is provided as link in the Signing Client in addition to the GDPR statement (max 2000 chars).

- **Signing Client online help URL**

`client.signing.general.onlinehelp.url`

The URL for the Signing Client online help. See also

`client.signing.general.onlinehelp.visible`

Default value: https://docshield.tungstenautomation.com/SD/en_US/3.4.0-rq5j77n6cd/help/StandardSigningDocuments/index.html

- **Enable Signing Client online help**

`client.signing.general.onlinehelp.visible`

If set to 'Off', the configured link for the online help will not be shown in the GUI. See also

`client.signing.general.onlinehelp.url`.

Default value: On

- **Enable persistent browser storage for images**

`client.signing.image.local.storage.enabled`

When a signer uploads a signature- or stamp image from the filesystem, the image is stored in the local browser's storage. If set to 'On', it is stored in the browser persistently. So the signer can re-use the image also in his following signing sessions. If set to 'Off', the image is removed from the local browser's storage when the signer closes the tab or the browser. In this case, the signer has to upload the image from the filesystem in his next signing session again.

Default value: On

- **Document auto-adjustment on mobile devices**

`client.signing.mobile.document.adjustment.enabled`

If set to 'On,' the signing document is scaled to the available screen width on mobile devices (tablets and phones) automatically when the document is opened in the Signing Client. If set to 'Off,' the document is displayed in its default resolution of 96 dpi. Please note that the document can only be scaled to a maximum resolution of 200 dpi. So for very large mobile screens, the document may not cover the whole screen width.

Default value: On

- **Default personal certificate signing appearances**

`client.signing.personalcertificate.signingappearances.default`

The comma separated list of signing appearances are used as default values for the 'Personal Certificate' signing mode option of a signature field. Valid signing appearances: HW (Handwritten), PH (Photo Capture), C2S (Click-to-Sign), IMG (Image), STAMP (Stamp), PLUGIN (Plug-in).

Default value: HW,PH,C2S,IMG,STAMP,PLUGIN

- **Allow easy restart of remote signing session**

`client.signing.remotesession.easy.restart.allowed`

If set to 'On', a signer can restart the remote signing session directly in the Signing Client view after it expires. If set to 'Off', the signer must open his invitation link in a browser again after the remote signing session expires.

Default value: On

- **Clear signature action availability**

`client.signing.signature.clear.available`

Defines if there is an opportunity for a signer to clear a signature field that is already signed. If set to 'Off', the action to clear a signed signature field is not available.

Default value: On

- **Default signature field signing mode options**

`client.signing.signaturefield.signingmodes.default`

The comma separated list of signing modes are used as default values for a signature field of a document. Valid signing modes: HW (Handwritten), PH (Photo Capture), C2S (Click-to-Sign), IMG (Image), STAMP (Stamp), TSP (Trusted Service Provider), PC (Personal Certificate). Note: If TSP is set as one of the values and the signer has no TSP information set, TSP will be ignored.

Default value: HW,PH,C2S,IMG,STAMP,TSP,PC

- **Show header**

`client.signing.view.general.header.visible`

Defines if the header is shown in the Signing Client.

Default value: On

- **Show footer**

`client.signing.view.general.footer.visible`

Defines if the footer is shown in the Signing Client.

Default value: On

- **Show wizard-steps**

`client.signing.view.general.wizardsteps.visible`

Defines if the wizard-steps are shown in the Signing Client.

Default value: On

- **Show instructions**

`client.signing.view.general.instructions.visible`

Defines if the instructions are shown in the Signing Client.

Default value: On

- **Show decline action**

`client.signing.view.general.decline.visible`

Defines if the decline action is visible in the Signing Client. If the action is not visible, the signer is not able to decline a signing session.

Default value: On

- **Show "Welcome" view**

`client.signing.view.welcome.visible`

Defines if the "Welcome" view is shown in the Signing Client. If set to 'Off', the "Welcome" view is skipped when no signer authentication (access code or external authentication) is used.

Default value: On

- **Show "In-person Signing" view**

`client.signing.view.inperson.visible`

Defines if the "In-person Signing" view is shown in the Signing Client. If set to 'Off', the "In-person Signing" view is skipped when there is only one signer in an in-person signing session.

Default value: On

- **Show "Review & Sign" view**

`client.signing.view.reviewsign.visible`

Defines if the "Review & Sign" view is shown in the Signing Client. If set to 'Off', the "Review & Sign" view is skipped when only one document is used in the signing session and no view-specific features are assigned to the signer, like TSP and supplemental documents.

Default value: On

- **Show progress in "Review & Sign" view**

`client.signing.view.reviewsign.progress.visible`

Defines if the progress bar is shown in the "Review & Sign" view.

Default value: On

- **Show download in "Review & Sign" view**

`client.signing.view.reviewsign.download.visible`

Defines if the download actions are shown in the "Review & Sign" view.

Default value: On

- **Show "Resume later" action in "Review & Sign" view**

`client.signing.view.reviewsign.resume_later.visible`

Defines if the "Resume later" action is shown in the "Review & Sign" view.

Default value: On

- **The external finish URL (remote signing session)**

`client.signing.view.finish.url`

The custom URL which is called when a signing session is finished by the remote signer.

It can be configured by placeholders for account ID, package ID and signer ID that are automatically replaced by data. Example: `http://www.myserver.com/mypage.html?accountid=$ACCOUNT_ID&packageid=$PACKAGE_ID&signerid=$SIGNER_ID`. If no URL is provided the default finish page is displayed in the Signing Client.

- **The external resume later URL (remote signing session)**

`client.signing.view.resume_later.url`

The custom URL which is called when a signing session is resumed for the latter by the remote signer. It can be configured by placeholders for account ID, package ID and signer ID that are automatically replaced by data. Example: `http://www.myserver.com/mypage.html?accountid=`

`$ACCOUNT_ID&packageid=$PACKAGE_ID&signerid=$SIGNER_ID`. If no URL is provided, the default finish page is displayed in the Signing Client.

- **The expired session URL (remote signing session)**

`client.signing.view.session_expired.url`

The custom URL which is called when a signing session of a remote signer expired. It can be configured by placeholders for account ID, package ID and signer ID that are automatically replaced by data. Example: `http://www.myserver.com/mypage.html?accountid=$ACCOUNT_ID&packageid=$PACKAGE_ID&signerid=$SIGNER_ID`). If no URL is provided, the default session expired page is displayed in the Signing Client.

- **The external end URL (in-person signing session)**

`client.signing.view.end.url.inperson`

The custom URL is called when an in-person signing session ends regularly. It regularly ends when either the signing host clicks the end session button on the "In-person Signing" view or the only signer clicks the finish button on the "Review & Sign" view, and the "In-person Signing" view is disabled. It can be configured by placeholders for account ID, package ID, and signer IDs, automatically replaced by data. The replaced data contains no signer IDs if the final URL exceeds 2083 characters. Example: `http://www.myserver.com/mypage.html?accountid=$ACCOUNT_ID&packageid=$PACKAGE_ID&finished_signerids=$FINISHED_SIGNER_IDS&unfinished_signerids=$UNFINISHED_SIGNER_IDS`. If no URL is provided the default end page is displayed in the Signing Client.

- **The expired session URL (in-person signing session)**

`client.signing.view.session_expired.url.inperson`

The custom URL is called when an in-person signing session expires. It can be configured by placeholders for account ID, package ID and signer IDs that are automatically replaced by data. The replaced data contains no signer IDs if the final URL exceeds 2083 characters. Example: `http://www.myserver.com/mypage.html?accountid=$ACCOUNT_ID&packageid=$PACKAGE_ID&finished_signerids=$FINISHED_SIGNER_IDS&unfinished_signerids=$UNFINISHED_SIGNER_IDS`. If no URL is provided, the default session expired page is displayed in the Signing Client.

Manage Client related settings

- **Signer names client colors**

`client.general.signer.colors`

The client displays each signer in a specific color. The setting contains a list of hexadecimal RGB colors (`#RRGGBB`) separated by commas. If there are more signers than colors, the colors are repeated.

Default value:

`#1c64b4,#00c425,#c4202f,#ca5602,#884fad,#6ad0f4,#00e580,#e57b8b,#f7db61,#c7a3da`

- **Skip the landing page**

`client.general.skip.landing`

Skips the initial landing page displayed before the login form.

Default value: On

- **Word to pdf document fonts directory**

`client.manage.document.word-pdf.font-directory`

Path to the fonts directory used to convert the signing package document from Word to PDF in the Manage Client. The directory defined should be a valid path and should contain the valid

required font. If a fonts directory is defined here, it will override the systems and the default fonts.

Default value: `SIGNDOC_HOME/fonts`

- **Manage Client online help URL**

`client.manage.general.onlinehelp.url`

The URL for the Manage Client online help. See also

`client.manage.general.onlinehelp.visible`

Default value: `https://docshield.tungstenautomation.com/SD/en_US/3.4.0-rq5j77n6cd/help/Standard/index.html`

- **Enable Manage Client online help**

`client.manage.general.onlinehelp.visible`

If set to 'Off', the configured link for the online help will not be shown in the GUI. See also

`client.manage.general.onlinehelp.url`.

Default value: On

- **Default signing package expiry date**

`client.manage.package.expiration`

The default number of days until a signing package expires. The value 0 means that a signing package never expires. The maximum supported value is 365 days.

Default value: 0

- **Recipients must be selected**

`client.manage.restrict.recipients.input`

If set to 'On', recipients cannot be entered manually, but must be selected from a list.

Default value: Off

- **Setting to display the administration center link**

`client.manage.show.admincenter.link`

If set 'on', the administration center link is shown to the user on the login page of the manage client else not shown.

Default value: On

- **Takeover attributes of selected signer**

`client.manage.signer.autocomplete.takeover`

Signer attributes are adopted from the selected signer after autocomplete search.

Default value: On

- **Generate first and last name proposal**

`client.manage.signer.name.proposal`

Defines whether the client fills the first name and last name fields with proposals derived from the signer name if external authentication is selected as authentication method for a signer.

Default value: On

Administration Client related settings

- **Administration Center online help URL**

`client.admin.general.onlinehelp.url`

The URL for the Administration Center online help. See also:

`client.admin.general.onlinehelp.visible`.

Default value: https://docshield.tungstenautomation.com/SD/en_US/3.4.0-rq5j77n6cd/help/StandardAdministrationCenter/index.html

- **Enable Administration Center online help**

`client.admin.general.onlinehelp.visible`

If set to 'Off', the configured link for the online help will not be shown in the GUI. See also:

`client.admin.general.onlinehelp.url`

Default value: On

Advanced signing settings

This section lists the advanced settings related to the signing process.

- **2FA access code length**

`cirrus.security.2fa.accesscode.length`

The length of the generated random access code for the 2-factor authentication.

Default value: 6

- **The external authentication provider name**

`cirrus.security.external.authentication.name`

The external authentication provider name. This name is displayed in the Manage Client as well as in the Signing Client as identification for the implemented external authentication service.

- **Use Google as an authentication provider**

`cirrus.security.external.authentication.oauth.google.enabled`

If set to 'On', a package creator can assign Google as one of the authentication providers for a recipient, provided Google is registered in the SignDoc application.

Default value: On

- **Use Microsoft as an authentication provider**

`cirrus.security.external.authentication.oauth.microsoft.enabled`

If set to 'On', a package creator can assign Microsoft as one of the authentication providers for a recipient, provided Microsoft is registered in the SignDoc application.

Default value: On

- **Validates recipient's email for OAuth authentication**

`cirrus.security.external.authentication.oauth.validate.recipient.email`

If set to 'On', the recipient's email is validated against the email they used for OAuth authentication using Google or Microsoft. Authentication fails if the recipient attempts to authenticate using a different email. If set to 'Off', recipients can authenticate using any valid email address from the respective providers.

Default value: Off

- **The external authentication service shared secret**

`cirrus.security.external.authentication.sharedsecret`

The external authentication service shared secret used to authenticate the system REST calls.

- **The external authentication service URL**

`cirrus.security.external.authentication.url`

The custom authentication service application URL.

- **Variable part of the application configuration shared secret**

`cirrus.security.ksd_appconf_shasec`

The application configuration shared secret is used to encrypt client to server communication. This setting lets you personalize the encryption.

- **Display text field values in audit trail**

`client.signing.audittrail.textfield.value.enabled`

If set to 'On', the text field data is displayed in the audit trail.

Default value: On

- **Preferred signature field overlay**

`client.signing.overlay.display.prefer`

Per default, an overlay image for a particular signing method of a signature field is displayed if the signing method is required for the signature field. This setting allows the display of the overlay image of a particular signing method also if this signing method is optional for the signature field. Currently, only 'TSP', which stands for the 'TSP signature' signing method, is supported as value.

- **Clear signature action availability**

`client.signing.signature.clear.available`

Defines if there is an opportunity for a signer to clear a signature field that is already signed. If set to 'Off', the action to clear a signed signature field is not available.

Default value: On

- **Signature image maximum size**

`client.signing.signature.image.size.max`

The maximum size (in KB) of an uploaded signature image which can be used for signing. The allowed range is between 1 and 1000 (KB).

Default value: 500

- **Signature (certificate) type**

`client.signing.tsp.signature.type`

Signature (certificate) type which is needed for signing (BASIC, ADVANCED or QUALIFIED).

Default value: BASIC

- **Enable Device Connector in the Signing Client**

`client.signing.deviceconnector.enable`

If set to 'On', the device connector will be enabled. See also `cirrus.signing.csp.enable` and `cirrus.signing.csp.value`.

Default value: On

- **Device Connector URL**

`client.signing.deviceconnector.url`

The URL of the Device Connector server.

Default value: `http://localhost:6613`

- **Encrypt Device Connector communication**

`client.signing.deviceconnector.encrypted`

Specifies if the communication with the Device Connector server is encrypted.

Default value: On

- **Require Device Connector on desktop devices**

`client.signing.deviceconnector.required`

Specifies if the Device Connector and a signature pad are required to capture signatures from the signers. If set to 'On', signing on Windows or Linux desktop devices requires a signature pad. If set to 'Off', signing is also possible in the browser when no signature pad is available. Note that on all other devices, signing is always done in the browser.

Default value: Off

AI (experimental)

This section lists the AI related settings.

Tasks > Document description

- **Enable manual AI document description**

`signdoc.ai.tasks.generate.document.description.enabled`

If on, SignDoc allows to generate a document description using an AI provider, after a document is uploaded.

Default value: Off

- **Enable automatic AI document description on upload**

`signdoc.ai.tasks.generate.document.description.auto`

If on, SignDoc will generate a document description on document upload using an AI provider.

Default value: Off

- **Number of words for document description**

`signdoc.ai.tasks.generate.document.description.words`

The number of words SignDoc will use to generate a document description using an AI provider. For a very large document, the AI generated description may not match the total number of words defined here and depends on the underlying AI model.

Default value: 50

- **Language selection for the document description**

`signdoc.ai.tasks.generate.document.description.language.source`

Defines how the language for the document description is derived. USER means the SignDoc user's language, DOCUMENT means the main language of the document, FIXED will use the value of `signdoc.ai.generate.document.description.language.fixed`. Valid sources: USER,DOCUMENT,FIXED

Default value: USER

- **The fixed language setting**

`signdoc.ai.tasks.generate.document.description.language.fixed`

This language (use the English name of the language) will be used to generate the document description, if `signdoc.ai.generate.document.description.language.source` is set to FIXED. Valid languages: Any language the AI model is capable of. Example: German, French, Japanese.

Default value: English

AI providers > Azure OpenAI

- **Enable the Azure OpenAI provider**

`signdoc.ai.providers.azure_openai.enabled`

If set to 'On' the Azure OpenAI provider is enabled.

Default value: Off

- **Azure OpenAI API key**

`signdoc.ai.providers.azure_openai.api_key`

The Azure OpenAI API key

- **Azure OpenAI endpoint**

`signdoc.ai.providers.azure_openai.endpoint`

The Azure OpenAI endpoint

- **Azure OpenAI deployment**

`signdoc.ai.providers.azure_openai.deployment`

Name of the Azure OpenAI deployment

Default value: gpt-35-turbo

- **Order of the Azure OpenAI provider**

`signdoc.ai.providers.azure_openai.order`

If multiple AI Providers are enabled the order attribute defines which provider has precedence. Precedence is defined by value. 0 has the highest priority. Larger values have lower priority.

Default value: 0

AI providers > OpenAI

- **Enable the OpenAI provider**

`signdoc.ai.providers.openai.enabled`

If set to 'On' the OpenAI provider is enabled.

Default value: Off

- **OpenAI API token**

`signdoc.ai.providers.openai.token`

The OpenAI API token. If this token is not set, this provider will not work.

- **Open AI model**

`signdoc.ai.providers.openai.model`

The OpenAI model to use.

Default value: gpt-4o

- **Order of the OpenAI provider**

`signdoc.ai.providers.openai.order`

If multiple AI Providers are enabled the order attribute defines which provider has precedence. Precedence is defined by value. 0 has the highest priority. Larger values have lower priority.

Default value: 0

System

This section lists the system settings.

General system settings

- **Configuration cache check time**

`cirrus.config.cache_check_period`

The time interval in milliseconds to check if the configuration cache has to be updated.

Default value: 30000

- **List of configurable languages**

`cirrus.config.locales`

The comma separated list of locale strings for storing and retrieving the locale-specific configuration values, such as email subject and body. A locale string must be a valid IETF BCP 47 language tag. See <https://www.rfc-editor.org/info/bcp47>. Example: en

Default value: en,de,fr,nl,it,es,pt-BR,ja,zh-CN

- **Logo on the final audit trail document**

`cirrus.general.audittrail.finaldocument.logo`

The logo on the final signing package audit trail document. If no logo is provided, default Tungsten logo is used instead. The supported image types for the logo are BMP, JPEG, TIFF, GIF, PNG, EMF, and WMF. The height and width of the logo image are adjusted to maintain the aspect ratio of the original image width to the image rectangle width on the page. However, if the resulting image height exceeds 80 pixels, the image will be compressed further to fit within the specified height limit.

Default value: TungstenLogo.png

- **Audit trails language**

`cirrus.general.audittrail.locale`

Language to be used for the audit trails. It must be a valid IETF BCP 47 language tag. See <https://www.rfc-editor.org/info/bcp47>. Example: en for English or pt-BR for Brazilian Portuguese.

Default value: en

Example: en for English or pt-BR for Brazilian Portuguese.

- **Controls the SignDoc Assistant notifications**

`cirrus.general.enable.assistant.app.notification`

If set to 'On', a notification will be sent to a device(s) registered with the signer's email.

Default value: Off

- **Controls the Microsoft Teams chat notifications**

`cirrus.general.enable.microsoftteams.notification`

If set to 'On', a notification will be sent to the signer's Microsoft Teams account registered with the signer's email.

Default value: Off

- **Customized localization data for the audit trails**

`cirrus.general.localization.audittrail`

User defined customized localization for the audit trails. The data expected is a valid key value pair such as key=value. The user can provide a complete set or a subset for the localization of a specific language.

Default value: audittrail.properties

- **Customized localization data for the Manage Client**

`cirrus.general.localization.client.manage`

User defined customized localization for the Manage Client. The data expected is a well-formed json. The user can provide a complete set or a subset for the localization of a specific language.


Default value: mc_language.json

- **Customized localization data for the Signing Client**

`cirrus.general.localization.client.signing`

User defined customized localization for the Signing Client. The data expected is a well-formed json. The user can provide a complete set or a subset for the localization of a specific language.

Default value: `sc_language.json`

 `cirrus.general.localization.*`, if the structure of the localization messages is changed between the releases, the customer should adapt to the new changes manually, also layout changes are not in the scope of these configurations. Message keys that are not included in the original localization files or are not in the correct tree structure will be ignored.

The final audit trail message is based on the language defined in the configuration `cirrus.general.audittrail.locale` at the time of writing the audit trail message. Hence if the language is changed when a signing package is in progress, the audit trail for that signing package will contain messages in mixed languages. This also affects the final PDF document for the signing package.

- **Customized localization data for Microsoft Teams chat notifications**

`cirrus.general.localization.microsoftteams.messages`

User defined customized localization for Microsoft Teams chat notifications. The data expected is a well-formed json. The user can provide a complete set or a subset for the localization of a specific language.

Default value: `bot_language.json`

- **Prefer officially supported languages by Tungsten**

`cirrus.general.localization.prefer.base.languages`

Tungsten maintains a list of officially supported GUI languages. If set to 'On' (default) and one of the accepted languages in the browser configuration is an officially supported Tungsten language, that language will be preferred over other languages in the user's language list. If set to 'Off', the browser will always try to use the user's most preferred language. This can lead to an English UI if there is no language file installed for the requested language.

Default value: On

- **Retention period for statistics data**

`cirrus.general.statistics.retention`

Retention period in days for statistics data. All the statistics data until current date minus this period (00:00 hours) will be deleted from the database and cannot be retrieved.

Default value: 730

- **Enable TotalAgility licensing**

`cirrus.general.totalagilitylicensing.enable`

If set to 'On' SignDoc will manage licenses through the TotalAgility License Server. Turning 'Off' will switch the user back to use SignDoc licenses.

Default value: Off

- **TotalAgility License Server API URL**

`cirrus.general.totalagilitylicensing.api.url`

TotalAgility License Server API URL to connect to TotalAgility License Server for license management.

- **TotalAgility License Server API Key**

```
cirrus.general.totalagilitylicensing.api.key
```

Unique identifier (API key) assigned to the TotalAgility License Server, enables secure communication and authentication with SignDoc.

- **State change query interval (minutes)**

```
cirrus.statechange.polling.interval
```

Defines the database query interval in minutes for signer- and signing package tables for pending state change.

Default value: 45

- **Certificate expiration warning threshold (days)**

```
client.account.certificate.expiry.warn.threshold
```

The threshold in days when the client will start warning of expiring certificate (1-365).

Default value: 20

- **License expiration warning threshold (days)**

```
client.account.license.expiry.warn.threshold
```


The threshold in days when the client will start warning on expiring account licenses (1-365).

Default value: 20

Single Sign-on settings (SSO)

In SignDoc Standard the SSO configuration is split into 2 parts:

- General SSO settings.
The general settings are described in the SSO section of the account configuration.
- SAML-related settings.
The SAML related settings, including descriptions, are available in the SAML section of the account configuration.

 Customers no longer need SignDoc Authentication Module (SAM), single sign-on is integrated in SignDoc as part of the SignDoc 3.4.0 release.

- **Automatically login with Single Sign-on**

```
cirrus.sso.autologin
```

If the context URL is opened in the browser, SignDoc tries to authenticate the user automatically by querying the configured Single Sign-on authentication module.

Default value: Off

- **Create new users**

```
cirrus.sso.create.user
```

Automatically create new Single Sign-on users in SignDoc.

Default value: On

- **Default account ID**

```
cirrus.sso.create.user.account
```

The default account ID to use for Single Sign-on user creation/authentication when account information is missing.

- **Update user attributes**

```
cirrus.sso.create.user.update_attributes
```


If set to 'On', Single Sign-on user attributes will be synced with information defined in the IDP. Supported attributes: email, name, roles, team memberships.

Default value: Off

- **Creation of new users with roles**

`cirrus.sso.create.user.with_roles`

If set to 'On', newly created Single Sign-on users can be assigned more roles than USER. By default, auto-created Sign-on users have role USER.

Default value: Off

- **Creation of new users with teams**

`cirrus.sso.create.user.with_teams`

If set to 'On', newly created Single Sign-on users can be assigned teams. By default, auto-created Sign-on users have no team membership.

Default value: Off

- **Sanitize external user ID**

`cirrus.sso.sanitize.userid`

Sanitize external user ID for automatically created Single Sign-on users.

Default value: On

REST API settings

- **Time to live for an in-person signing session authentication token**

`cirrus.rest.authentication.signing.token.ttl.common`

The time to live in minutes for an in-person signing session token (X-S-AUTH-TOKEN) of the Signing Client.

Default value: 5

- **Time to live for a remote signing session authentication token**

`cirrus.rest.authentication.signing.token.ttl.remote`

The time to live in minutes for a remote signing session token (X-S-AUTH-TOKEN) of the Signing Client.

Default value: 5

- **Time to live for a Manage Client authentication token**

`cirrus.rest.authentication.token.leasetime`

Defines the time to live in minutes for a Manage Client authentication token (X-AUTH-TOKEN).

Default value: 240

- **Activate Content-Security-Policy header**

`cirrus.rest.csp.enable`

If set to 'On', the value of `cirrus.rest.csp.value` will be used as Content-Security-Policy header in HTTP responses.

Default value: On

- **Content-Security-Policy header value**

`cirrus.rest.csp.value`

Sets the value of the Content-Security-Policy header. Only used if `cirrus.rest.csp.enable` is set to 'On'.

Default value: default-src 'none'; style-src 'self' 'unsafe-inline'; script-src 'self'; img-src 'self' data: blob: https://signdoc-assistant.azurewebsites.net; frame-src 'self'; connect-src 'self'; font-src 'self'; media-src 'self'; object-src 'none'; form-action 'self'; frame-ancestors 'self' https://teams.microsoft.com;

- **Allow to use the field name to reference document fields**

```
cirrus.rest.document.field.use_name_as_id_fallback
```

Instead of having to use field ID values in the JSON data of a POST /v8/package request to reference document fields (text, checkbox, signature), it is possible to reference the fields also per field name if this configuration setting is set to 'On'.

Default value: On

- **User authentication check time**

```
cirrus.rest.event.user.authenticated.request.authid_check_period
```

The time interval in seconds to check if the user was authenticated with another authentication ID. The verification is needed by the Manage Client for auto logoff.

Default value: 10

- **Default auto-prepare option for documents**

```
cirrus.rest.expresspackage.auto-prepare
```

If set to 'On', creates one signature field for each signer in every document of the express signing package.

Default value: Off

- **Default delete-existing option for signing package**

```
cirrus.rest.expresspackage.delete-existing
```

Deletes existing package with the same ID before creating a new express signing package.

Default value: Off

- **Default 'required' flag for signature fields**

```
cirrus.rest.expresspackage.signaturefield.required.default
```

Default setting for the required flag of signature fields in a express signing package when no signature field is marked required.

Default value: On

- **Default signer authentication method**

```
cirrus.rest.expresspackage.signer.authentication.default
```

The default authentication method that is used for the signer when creating an express signing package.

Default value: Authentication not required.

- **Signer default name**

```
cirrus.rest.expresspackage.signer.default.name
```

Default name for signer in the express signing package if signer name is not specified in the request.

Default value: Signer

- **Validate HTML input and reject, if invalid or dangerous**

```
cirrus.rest.html_input.validate
```

If HTML fragments contain invalid or dangerous constructs, the data will be rejected.

Default value: On

- **Sanitize HTML input, if invalid or dangerous**

`cirrus.rest.html_output.sanitize`

If HTML fragments contain invalid or dangerous constructs, the data will be sanitized.

Default value: On

- **General REST result set size limit**

`cirrus.rest.resultset.size.max.general`

General size limit of result sets for lists which are retrieved via REST API. The general settings can be overwritten by resource specific settings.

Default value: 500

- **Package REST result set size limit**

`cirrus.rest.resultset.size.max.packages`

Size limit of result sets for package lists which are retrieved via REST API. If this configuration value is not set, then the general setting `cirrus.rest.resultset.size.max.general` is used for result set limitation.

- **Signer REST result set size limit**

`cirrus.rest.resultset.size.max.signers`

Size limit of result sets for signer lists which are retrieved via REST API. If this configuration value is not set, then the general setting `cirrus.rest.resultset.size.max.general` is used for result set limitation.

- **Signer REST result set size limit if a search criteria is being used**

`cirrus.rest.resultset.size.max.signers.autocomplete`

Size limit of result sets for signer lists which are retrieved via REST API in case one of the search criteria is being used. This is the case for the client autocomplete function.

Default value: 20

- **User REST result set size limit**

`cirrus.rest.resultset.size.max.users`

Size limit of result sets for user lists which are retrieved via REST API. If this configuration value is not set, then the general setting `cirrus.rest.resultset.size.max.general` is used for result set limitation.

- **Include the locking signer's email when a signing package is locked**

`cirrus.rest.signingsession.status.return.signer.email`

When this setting is 'On' and a signing package is locked the GET `signingsession/status` endpoint will include the email of the signer who is currently locking the signing package.

Default value: On

- **Include the locking signer's name when a signing package is locked**

`cirrus.rest.signingsession.status.return.signer.name`

When this setting is 'On' and a signing package is locked, the GET `signingsession/status` endpoint will include the name of the signer who is currently locking the signing package.

Default value: On

- **Require password to change a user's email address**

`cirrus.rest.user.email_change.require_password`

If set to 'On', the authenticated user must also supply the own password to change any user's email address.

Default value: On

- **Enable Content-Security-Policy header for the Signing Client**

`client.signing.csp.enable`

If set to 'On', the client will use the Content-Security-Policy header as defined by `cirrus.signing.csp.value`.

Default value: On

- **Content-Security-Policy header of the Signing Client**

`client.signing.csp.value`

Content-Security-Policy header of the Signing Client. Only used, if `cirrus.signing.csp.enable` is set to 'On' for an account.

Default value: default-src 'none'; style-src 'self' 'unsafe-inline'; script-src 'self'; img-src 'self' data: blob: https://signdoc-assistant.azurewebsites.net; frame-src 'self'; connect-src 'self'; font-src 'self'; media-src 'self'; object-src 'none'; form-action 'self'; frame-ancestors 'self' https://teams.microsoft.com;

- **Automatic field masking**

`cirrus.rest.field.masking.automatic`

If set to 'On', fields that cannot be edited by the current signer are masked in the Signing Client and cannot be read by the current signer. Fields that are not assigned to a particular signer or reviewer can be read by all signers or reviewers.

Default value: Off

- **Automatic field masking for reviewers**

`cirrus.rest.field.masking.viewer`

If set to 'Off', a reviewer will see the values of all document fields when "Automatic field masking" is turned 'On'. If set to 'On', the reviewer will only see values of fields assigned to 'Any'.

Default value: Off

- **Color of the masking box**

`cirrus.rest.field.masking.color.box`

If set, the masking box will be displayed in the specified color in RGB Hex notation e.g. E0E0E0 is light grey, FFFF00 is yellow.

Default value: 404040

- **Text displayed on masked fields**

`cirrus.rest.field.masking.text`

If set, this text will be displayed on masked fields, to give the signer more information about the masked field. Remark: If the entered text is too long, it might not be readable or not be even visible.

Default value: Confidential

- **Color of the displayed text**

`cirrus.rest.field.masking.color.text`

If set, this text will be displayed in the specified color in RGB Hex notation e.g. 404040 is dark grey, 000000 is black.

Default value: E0E0E0

- **Font of the displayed text**

`cirrus.rest.field.masking.font.text`

If set, the specified font will be used. Remark: The specified font must be a TrueType font.

Default value: `masking-font.ttf`

SAML settings

- **SAML IDP metadata as URL**

`cirrus.sso.saml.idp.metadata.url`

The IDP metadata is usually available via a public URL which can be set here. Setting the URL has the advantage that the metadata is retrieved on every authentication request and is so updated automatically. This setting is ignored if `cirrus.sso.saml.idp.metadata.xml` is also set.

Default value: `https://dev-70867513.okta.com/app/exk91jsoisJjrZbqg5d7/sso/saml/metadata`

- **SAML IDP metadata as XML**

`cirrus.sso.saml.idp.metadata.xml`

The IDP metadata is in XML format and can be set here. Setting the XML has the advantage that it is the quickest approach since it does not dynamically retrieve the IDP metadata for every request over the network. This setting will overwrite `cirrus.sso.saml.idp.metadata.url`.

- **Email attributes from SAML response**

`cirrus.sso.saml.idp.attribute.email`

A list of SAML attributes for setting the email of a SignDoc user. `PRINCIPAL_NAME` is a special placeholder since it can be a user ID or email, depending on the configuration of the IDP. The attributes are evaluated from top to bottom. The first match wins.

Default value: `PRINCIPAL_NAME signdoc_email email user.email`

- **Login user identity attributes from SAML response**

`cirrus.sso.saml.idp.attribute.login`

A list of SAML attributes to use for defining the user ID attribute of a SignDoc user. `PRINCIPAL_NAME` is a special placeholder since it can be a user ID or email, depending on the configuration of the IDP. The attributes are evaluated from top to bottom. The first match wins. This setting is only used if `cirrus.sso.create.user` is set to 'On'.

Default value: `PRINCIPAL_NAME signdoc_login login user.login http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name`

- **Name attributes from SAML response**

`cirrus.sso.saml.idp.attribute.name`

A list of SAML attributes for setting the user name of a SignDoc user. The attributes are evaluated from top to bottom. The first match wins. This setting is only used if `cirrus.sso.create.user` is set to 'On'.

Default value: `signdoc_name name displayName user.displayName http://schemas.microsoft.com/identity/claims/displayname`

- **First name attributes from SAML response**

`cirrus.sso.saml.idp.attribute.first_name`

A list of SAML attributes for setting the first name of a SignDoc user. This setting is only used if `cirrus.sso.saml.idp.attribute.name` returns no result. The attributes are evaluated from top to bottom. The first match wins. This setting is only used if `cirrus.sso.create.user` is set to 'On'.

Default value: `signdoc_firstname` `firstname` `user.firstName` `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname`

- **Last name attributes from SAML response**

`cirrus.sso.saml.idp.attribute.last_name`

A list of SAML attributes for setting the last name of a SignDoc user. This setting is only used if `cirrus.sso.saml.idp.attribute.name` returns no result. The attributes are evaluated from top to bottom. The first match wins. This setting is only used if `cirrus.sso.create.user` is set to 'On'.

Default value: `signdoc_lastname` `lastname` `user.lastName` `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname`

- **Roles attributes from SAML response**

`cirrus.sso.saml.idp.attribute.roles`

A list of SAML attributes for setting the user's SignDoc roles. Valid roles are: `user`, `teammgr`, `admin`. The roles must be specified in a comma separated string. The attributes are evaluated from top to bottom. The first match wins. This setting is only used if `cirrus.sso.create.user` or `cirrus.sso.create.user.update_attributes` is set to 'On' and `cirrus.sso.create.user.with_roles` is set to 'On'.

Default value: `signdoc_roles`

- **Teams attributes from SAML response**

`cirrus.sso.saml.idp.attribute.teams`

A list of SAML attributes for setting the user's SignDoc team memberships. The teams must be specified in a comma separated string. The attributes are evaluated from top to bottom. The first match wins. This setting is only used if `cirrus.sso.create.user` or `cirrus.sso.create.user.update_attributes` is set to 'On' and `cirrus.sso.create.user.with_teams` is set to 'On'.


Default value: `signdoc_teams`

- **Display SAML authentication results**

`cirrus.sso.saml.display_result`

Instead of redirecting to SignDoc, display a page listing all SAML assertion's SAML attributes. This is useful to find out how to map user-related attributes like user name or email. This information can be used for the config options matching the pattern `cirrus.sso.saml.idp.attribute.*`.

Default value: Off

 SAML configuration instructions will be displayed in the account **Status information** tab by default.

Logging

- **SignDoc log level**

`signdoc.logger.level`

Sets the base log level for all SignDoc specific components. A level more verbose than INFO (i.e., `DEBUG` | `FINE` | `FINER` | `FINEST` | `TRACE` | `ALL`) might impact the performance of the system. Valid log levels: `OFF` | `FATAL` | `SEVERE` | `ERROR` | `WARNING` | `WARN` | `INFO` | `CONFIG` | `DEBUG` | `FINE` | `FINER` | `FINEST` | `TRACE` | `ALL`

Default value: INFO

- **Log level for non SignDoc components**

```
signdoc.logger.other_levels
```

Sets the base log level for all non SignDoc components like e.g. Spring and Tomcat. A level more verbose than INFO (i.e., DEBUG | FINE | FINER | FINEST | TRACE | ALL) might impact the performance of the system. Valid log levels: OFF | FATAL | SEVERE | ERROR | WARNING | WARN | INFO | CONFIG | DEBUG | FINE | FINER | FINEST | TRACE | ALL

Default value: INFO

- **Custom log levels**

```
signdoc.logger.custom_levels
```

Provides customized log levels that overwrite the default log level defined by signdoc.logger.level. Use one line per logger. A leading # comments the line. Valid log levels: OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL. Syntax: `LOGGER_NAME=LOG_LEVEL`

Default value: #REST API exceptions

```
#de.softpro.cirrus.rest.common.controller.BaseRestController=FINE #REST API
requests #de.softpro.cirrus.web.helper.RequestLogFilter=FINER #Apache Tomcat logs
#org.apache.tomcat=INFO #org.apache.catalina=INFO
```

Documents and packages

This section lists the settings related to documents and packages.

Document

- **Source of the signer ID which is assigned to a signature (line) field**

```
cirrus.document.prepare.msword.signatureline.signerid.source
```

A Microsoft Word document can be uploaded to a signing package. The document can contain Word specific signature lines. A signature field is created in the converted PDF document for each signature line. In addition, a signer is assigned to the new created signature field. The signer is either already available in the package or it will be created automatically. If the value is 'field_name' then the signer ID is set from the signature line tag ID (which is also used for the new created signature field). If the value of this setting is 'signer_name' then the source of the ID for this signer is the 'Suggested signer' name from the Signature setup dialog (visible during insert signature line action). Note: The entered signer name must be conform to the allowed characters for an ID [a-zA-Z0-9_-] and must not contain spaces. Otherwise, a random UUID is set as signer ID. Allowed values for this configuration settings are either field_name or signer_name.

Default value: field_name

- **The maximum document size**

```
cirrus.document.prepare.size.max
```

The maximum size of a document that can be uploaded.

Default value: 10485760

- **Maximum number of files for a supplemental document type**

```
cirrus.document.prepare.supplemental.file.max-number
```

The maximum number of supplemental files that can be uploaded by a signer for one document type. Changing this number does not have an automatic impact on already configured document types or document type instances. This setting is only considered if you change or add a document type.

Default value: 25

- **Maximum number of supplemental document type instances**

`cirrus.document.prepare.supplemental.type-instance.max-number`

The maximum number of supplemental document type instances that can be created for a signer.

Default value: 25

- **Controls if documents are tested after upload**

`cirrus.document.prepare.validate.on.upload`

If set to 'On', documents lacking advanced compliance or having known vulnerabilities will be rejected.

Default value: On

- **The date pattern to use in Click-to-Sign signatures**

`cirrus.document.signing.c2s.date.format.pattern`

The pattern must be compatible with `SimpleDateFormat`. See also <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/SimpleDateFormat.html>. Examples: `yyyy-MM-dd HH:mm:ss`, `MM/dd/yyyy`, `MM/dd/yyyy h:mm a`, `dd.MM.yyyy`, `yyyy-MM-dd'T'HH:mm:ssZ`

Default value: `yyyy-MM-dd'T'HH:mm:ssZ`

- **The script font for Click-to-Sign signatures**

`cirrus.document.signing.c2s.font.script`

Click-to-Sign signatures need a "script font" that is used to render the signers's name. If the uploaded data is not usable it will be rejected.

The default Click-to-Sign script font covers only a small subset (Western-Latin) of Unicode. If other characters are entered, they will not show up in the resulting Click-to-Sign signature. To circumvent this, it is necessary to upload a font that contains all the required characters.

Default value: `c2s-font-script.ttf`

- **The text font for Click-to-Sign signatures**

`cirrus.document.signing.c2s.font.text`

Click-to-Sign signatures need a "text font" that is used to render the text data in a Click-to-Sign signature. If the uploaded data is not usable it will be rejected.

Default value: `c2s-font-text.ttf`

- **Logo text for Click-to-Sign signatures**

`cirrus.document.signing.c2s.logo.text`

Logo text that is displayed in Click-to-Sign signatures

No default value set.

- **The resolution in DPI for Click-to-Sign signatures**

`cirrus.document.signing.c2s.resolution.dpi`

Click-to-Sign signatures will be rendered in this resolution. Valid range: ≥ 72 and ≤ 600

Default value: 300

- **Certify documents sent to recipients**

`cirrus.document.signing.certify.final_artifacts`

Apply an additional certification signature to outgoing final PDF artifacts. If set to ALL, a signing package will use multiple digital signatures to certify all PDF documents before SignDoc will send them to the recipients. Valid options: All, NONE.

Default value: ALL

- **Defines if the attachments in the final email should be compressed**

`cirrus.document.signing.final-package.attachments.compress`

If set to 'On', the attachments in the final email sent to the signer(s) and the owner of the signing package will be compressed in a zip.

Default value: Off

- **Defines if the final email contains a link to download the documents**

`cirrus.document.signing.final-package.attachments.download`

If set to 'On', the final email sent to the signer(s) and the owner of the signing package will have a link to download the final documents and will not be attached in the email.

Default value: Off

- **Controls the attachments in the final email**

`cirrus.document.signing.final-package.container-document`

If set to 'On', the final email sent to the signer(s) and the owner of the signing package will have the container document with other documents embedded in it. Otherwise, the documents of the signing package along with the combined final audit trail are sent as multiple attachments.

Default value: Off

- **The cover page of the final document package**

`cirrus.document.signing.final-package.cover-document`

The cover page of the final document package can be customized for a specific language by uploading a flat PDF document. If a cover page is not available for a specific locale then the default cover page in English will be used. If the uploaded data is not usable it will be rejected.

Default value: cover-page.pdf

- **Font directory for the final document package**

`cirrus.document.signing.final-package.font-directory`

The customized font directory for the final document of the package. The directory defined should be a valid path and should contain the valid required font.

Default value: SIGNDOC_HOME/fonts

- **Font name for the final document package**

`cirrus.document.signing.final-package.font-name`

The font name defined should be either present in the font directory or the system font directory.

Default value: Noto Sans

- **Custom configuration for Device Connector plug-ins**

`cirrus.document.signing.personal.certificate.custom`

Add custom configuration for Device Connector plug-ins that might be used when signing.

Default value: no default set

- **Maximum size of supplemental documents**

`cirrus.document.signing.supplemental.file.max-size`

The maximum size (in kilobytes) of a single supplemental documents that can be uploaded by a signer.

Default value: 5000

- **Accepted supplemental documents file suffixes**

`cirrus.document.signing.supplemental.file.suffixes`

List of accepted file type extensions (file name suffix after period) for supplemental documents. Listed file type extensions must be separated by ',' and should not include any whitespace. By default the following file types are accepted: jpg,jpeg,png,doc,docx,pdf.

Default value: jpg,jpeg,png,doc,docx,pdf

- **The screen of the signature pad**

`cirrus.document.signing.tablet.screen`

Defines the layout and content of the screen when signing with a signature pad. If the uploaded data is not correct it will be rejected. Please note that Chinese and Japanese characters are currently not supported.

Default value: tablet-screen.xml

Package

- **Allowed MIME Types for uploaded documents**

`cirrus.package.allowed_mimetypes`

Specifies the MIME Types that SignDoc accepts for uploaded documents. Use one MIME Type per line.

Default value: application/pdf image/jpeg image/png image/tiff application/msword application/vnd.openxmlformats-officedocument.wordprocessingml.document

- **Enable delegate signing**

`cirrus.package.delegate.enabled`

Controls the delegation of a signing session for a recipient. If set to 'On', the recipient can delegate it's signing session to a new recipient. This flag will be ignored and is overridden if the signing package creator explicitly enables or disables delegate option for a recipient.

Default value: On

- **Controls if delegate signing is overwritable**

`cirrus.package.delegate.overwritable`

If set to 'On' the signing package creator can overwrite the default delegate signing session option else it is not allowed to make any changes.

Default value: On

- **Delete audit trail together with package**

`cirrus.package.delete.audittrail`

Defines whether all package related audit trail entries are deleted automatically if the package is removed.

Default value: On

- **Default value to enforce a signing method (configured on account level)**

`cirrus.package.enforceSigningMode.default`

Enforces a specific signing method for all the signature fields of a signer in a document of a signing package. If set to 'On' the signer is forced to sign the rest of the signature fields with the same signing mode it used to sign the first signature field. This flag will be ignored and is overridden if the package creator explicitly enables or disables this feature during package create/update.

Default value: Off

- **Defines if 'enforceSigningMode' flag default value is overwritable (configured on account level)**

`cirrus.package.enforceSigningMode.overwritable`

If set to 'On', the user can overwrite the 'enforceSigningMode' default value of the signing package. If set to 'Off', user is not allowed to make any changes and the default value is used.

Default value: On

- **Enable in-person signing**

`cirrus.package.inperson.enabled`

Controls the in-person signing. If set to 'On' in-person signing of a package is allowed else not allowed. This flag will be ignored and is overridden if the user explicitly enables or disables in-person signing during package create/update.

Default value: On

- **Controls if in-person signing option is overwritable**

`cirrus.package.inperson.overwritable`

If set to 'On' the user can overwrite the in-person signing option. If set to 'Off' user is not allowed to do any changes.

Default value: On

- **Enable signing package retention**

`cirrus.package.retention.enabled`

If switched to 'On', the signing packages are retained for the retention period defined others are deleted using a background process .

Default value: Off

- **Older than retention period**

`cirrus.package.retention.olderthan`

Defines the number of days the signing packages should be retained. Packages matching the retention package states that are older than this period will be deleted. The retention period calculation considers both the older than and the before date properties, the final period is the oldest date amongst these two.

Default value: 730

- **Before date retention period**

`cirrus.package.retention.beforedate`

Defines the date the signing packages should be retained. Packages matching the retention package states that are older than this period will be deleted. The before date must be specified in the format 'YYYY-MM-DD'. The retention period calculation considers both the older than and the before date properties, the final period is the oldest date amongst these two.

- **Signing packages retention states**

`cirrus.package.retention.states`

Comma separated signing package states, signing packages matching these states will be deleted based on the retention period defined. List of allowed states are [ALL_STATES,END_STATES,DRAFT,PREPARED,STARTED,COMPLETE,REJECTED,EXPIRED,CANCELED,ARCHIVED]. ALL_STATES will delete all packages in the retention period. END_STATES is synonym to [COMPLETE,REJECTED,EXPIRED,CANCELED,ARCHIVED]

Default value: END_STATES

Plug-ins

This section lists the settings related to plug-ins.

Plug-in implementations

This section contains the implemented plug-ins.

Enabled

This section contains the enabled plug-ins.

Configuration

In this section the enabled plug-ins can be configured.

General

- **Plug-in load list**

`plugin.loadlist`

The list of plug-ins to be loaded. Entries must be separated by ','.

- **Plug-in directory**

`plugin.directory`

The directory where plug-ins are located (in addition to the CLASSPATH).

- **Reload plug-ins at runtime**

`pluginss.reload`

If set to 'On', changes to `plugin.loadlist` are evaluated at runtime and do not need a service restart. It is recommended to only turn this setting 'On' when required and turn it 'Off' when no longer needed.

Default value: Off

- **Reject document on general document scan error**

`plugins.event.document_scan.reject_on_error`

If set to 'On', document scan errors that are not content related, will reject the uploaded document.

Default value: Off

Webhooks

This section lists the webhook related settings.

General

- **Enable webhooks**

`webhook.general.enabled`

If set to 'On', webhooks are active for the account.

Default value: Off

- **Send large data in webhook payload**

`webhook.general.event.post.blobs`

If set to 'On', the webhook request payload will also contain large data such as PDF documents.

Default value: Off

- **Send a REST Auth token to webhook receiver**

`webhook.general.event.post.auth`

If set to 'On', the webhook request headers will, when applicable for the event, contain a valid X-AUTH-TOKEN of the signing package owner that can be used for follow-up interaction with the REST API.

Default value: On

- **Webhook read timeout in seconds**

`webhook.general.timeout.read`

If a Webhook does not respond within this timeframe in seconds, an error will be generated.

Default value: 60

- **Webhook write timeout in seconds**

`webhook.general.timeout.write`

If set to 'On', webhooks are active for the account.

Default value: 60

- **Webhook connect timeout in seconds**

`webhook.general.timeout.connect`

If a Webhook does not connect within this timeframe in seconds, an error will be generated.

Default value: 10

Types

- **Enable the state_change webhook type**

`webhook.type.state_change.enabled`

If set to 'On', the state_change webhook type is active.

Default value: Off

- **Webhook URL of the state_change type**

`webhook.type.state_change.url`

The URL that consumes the state_change webhook type. If not set, the webhook type will not be processed.

- **State Change Webhook read timeout in seconds**

`webhook.type.state_change.timeout.read`

If a State Change Webhook does not respond within this timeframe in seconds, an error will be generated. This setting overwrites the general read timeout setting. If not set, the general read timeout setting will be used.

- **State Change Webhook write timeout in seconds**

`webhook.type.state_change.timeout.write`

If a State Change Webhook does not accept data within this timeframe in seconds, an error will be generated. This setting overwrites the general write timeout setting. If not set, the general write timeout setting will be used.

- **State Change Webhook connect timeout in seconds**

`webhook.type.state_change.timeout.connect`

If a State Change Webhook does not connect within this timeframe in seconds, an error will be generated. This setting overwrites the general connect timeout setting. If not set, the general connect timeout setting will be used.

- **Enable the tsp webhook type**

`webhook.type.tsp.enabled`

If set to 'On', the tsp webhook type is active.

Default value: Off

- **Webhook URL of the tsp type**

`webhook.type.tsp.url`

The URL that consumes the tsp webhook type. If not set, the webhook type will not be processed.

- **TSP Webhook read timeout in seconds**

`webhook.type.tsp.timeout.read`

If a TSP Webhook does not respond within this timeframe in seconds, an error will be generated. This setting overwrites the general read timeout setting. If not set, the general read timeout setting will be used.

- **TSP Webhook write timeout in seconds**

`webhook.type.tsp.timeout.write`

If a TSP Webhook does not accept data within this timeframe in seconds, an error will be generated. This setting overwrites the general write timeout setting. If not set, the general write timeout setting will be used.

- **TSP Webhook connect timeout in seconds**

`webhook.type.tsp.timeout.connect`

If a TSP Webhook does not connect within this timeframe in seconds, an error will be generated. This setting overwrites the general connect timeout setting. If not set, the general connect timeout setting will be used.

Chapter 4

Plug-ins

This chapter provides information on plug-in administration and development, including the handling of the available plug-ins.

Plug-in handling

SignDoc Standard supports the extension and/or customization of server logic via server side plug-ins. These event plug-ins are triggered by certain events on server side and enable the plug-ins to handle these events in an appropriate way.

There are 2 kinds of plug-ins:

- **Core plug-ins** are always loaded and are enabled by default for all accounts.
- **Custom plug-ins** must be explicitly loaded and enabled for the desired accounts before they can be used. These actions can be done at runtime and do usually not require a service restart.

How to implement a plug-in

The required resources to implement a plug-in can be found in the `signdoc_home/interfaces/plugins/` directory.

This directory contains the required components:

- Documentation
 - SignDoc Standard plug-in definitions:
`cirrus-plugin-definitions-VERSION-javadoc.zip`
 - Plug-in interface:
`spplugin-if-VERSION-javadoc.zip`
- Minimal compile time dependencies
 - `cirrus-plugin-definitions-VERSION.jar`
 - `spplugin-if-VERSION.jar`
 - `spplugin-fw-VERSION.jar`

See [Minimal SigningEvent implementation](#) and [Minimal SigningRSA implementation](#) for some basic examples of plug-ins.

How to deploy a plug-in

The `signdoc_home` directory of SignDoc Standard contains the `signdoc_home/plugins` directory where all plug-ins can be installed.

General directory structure

It is possible to organize multiple plug-ins in subdirectories of `signdoc_home/plugins/`. SignDoc Standard will create in each newly created directory 2 subdirectories: `classes/` and `lib/`.

The plug-in directory `default/` is treated in a special way. This directory is always present and has the highest priority amongst all plug-in directories. If unsure, plug-ins should be deployed in this `default/` directory.

```
signdoc_home/plugins/
|
|----- default/ (overrides classes of other plugin folders)
|
|     |
|     |----- classes/
|     |     |
|     |     |----- single classes/resources
|     |     |
|     |     |----- lib/
|     |     |     |
|     |     |     |----- jar files
|     |
|----- a_custom_plugin/ (custom plugin folders can be used to organise plugins)
|
|     |
|     |----- classes/
|     |     |
|     |     |----- single classes/resources
|     |     |
|     |     |----- lib/
|     |     |     |
|     |     |     |----- jar files
```

- `classes/`
This directory can hold single classes and resources and overrides the same classes of a `lib` directory in the same plug-in directory. The classes and resources must be organized in a directory structure that represent the package of a very class.
Example: The class `com.company.PluginClass` must be put in the directory `com/company/` to be recognized by SignDoc Standard.
- `lib/`
This directory can hold jar files.



- Classes in the `classes` directory override classes with the same class name in jar files (same behavior like war files).
- Classes in `plugin` directory cannot override classes that have already been loaded by the SignDoc Standard server application.
- The `default` plug-in directory has the highest priority. That is, a class deployed in the `default/` directory will always be preferred over a class with the same class name in a custom plug-in directory (such as `a_custom_plugin/`).
- The priority of the plug-in directories other than the `default/` plug-in directory is undefined. That is, if a class with the same classname is present in different directories it is undefined, which of these classes will be used.
- A plug-in must be deployed with all required dependencies.
- If a new plug-in is deployed, the files are immediately available for the SignDoc Standard application. A restart of the server application is not required.
- If plug-in directories and/or plug-in files are removed, the files are immediately unavailable for the SignDoc Standard application. A restart of the server application is not required.
- If the `default/` plug-in directory is deleted, it will be immediately recreated with empty `classes/` and `lib/` directories. All prior existing classes or resources will be immediately unavailable for the SignDoc Standard application.

Plug-in administration

Administer plug-ins in the Administration Center

Plug-ins can be administered and configured for all accounts in the SignDoc Standard Administration Center.

Tasks that can be done in the Administration Center are:

- Load or unload a plug-in.
Loading or unloading a plug-in can only be done in the Administration Center.
- Enable or disable a plug-in.
If a plug-in is enabled or disabled in the Administration Center, it will be used as default setting for all accounts.
- Assign a plug-in implementation to a specific event.
If a plug-in is assigned to a specific event in the Administration Center, it can be used as default/global implementation for all accounts.
- Configure a plug-in.
The plug-in configuration that is set in the Administration Center will be used as default/global configuration for all accounts.

Administer plug-ins in the Manage Client

Plug-ins can be configured by account administrators in the SignDoc Standard Manage Client. An account-specific configuration will override any global setting done in the Administration Center.

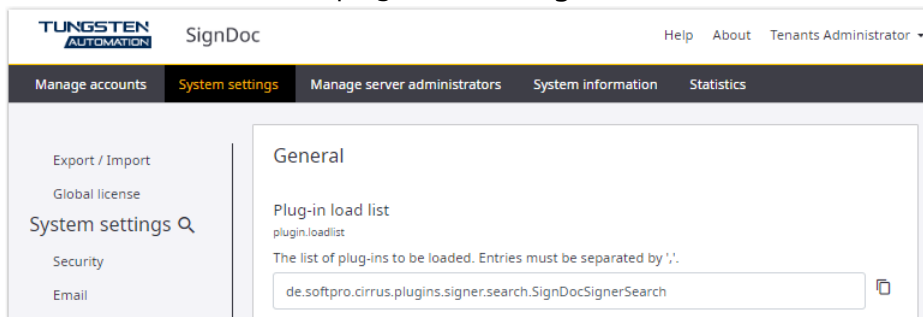
Tasks that can be done as account-specific administration are:

- Enable or disable a plug-in.
This overrides any default/global settings.
- Assign a plug-in implementation to a specific event.
This overrides any default/global settings.
- Configure a plug-in.
This overrides any default/global settings.

Load plug-ins in the Administration Center

i A plug-in can be loaded only in the Administration Center.

1. In the Administration Center, on the **System Settings** menu, click **Plug-ins > General**.
2. Add the class name of the plug-in to the **Plug-in load list** and save the settings.

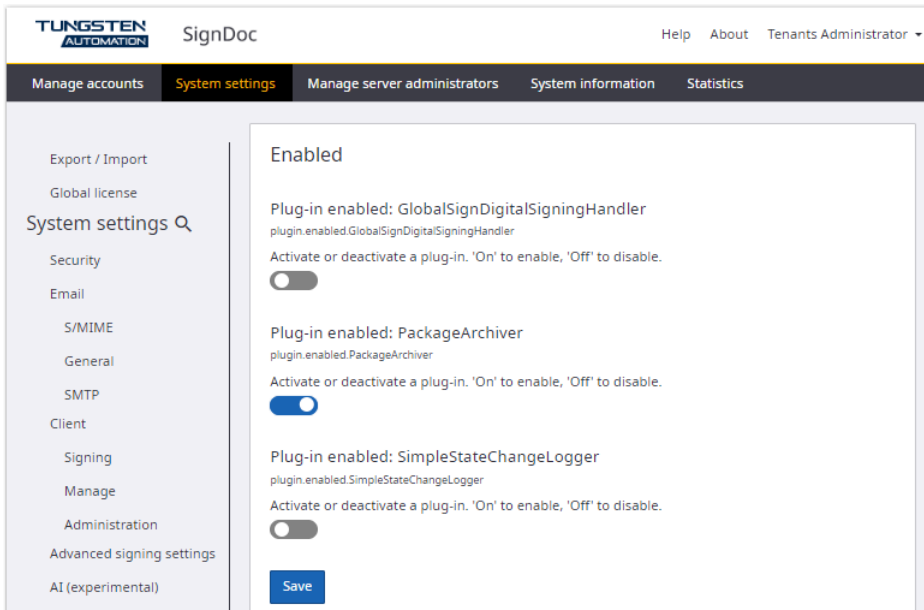


Enable or disable a plug-in

- i**
- A plug-in can be enabled or disabled in the Administration Center and in the account administration area of the Manage Client.
 - Before a plug-in can be enabled, it must be loaded.

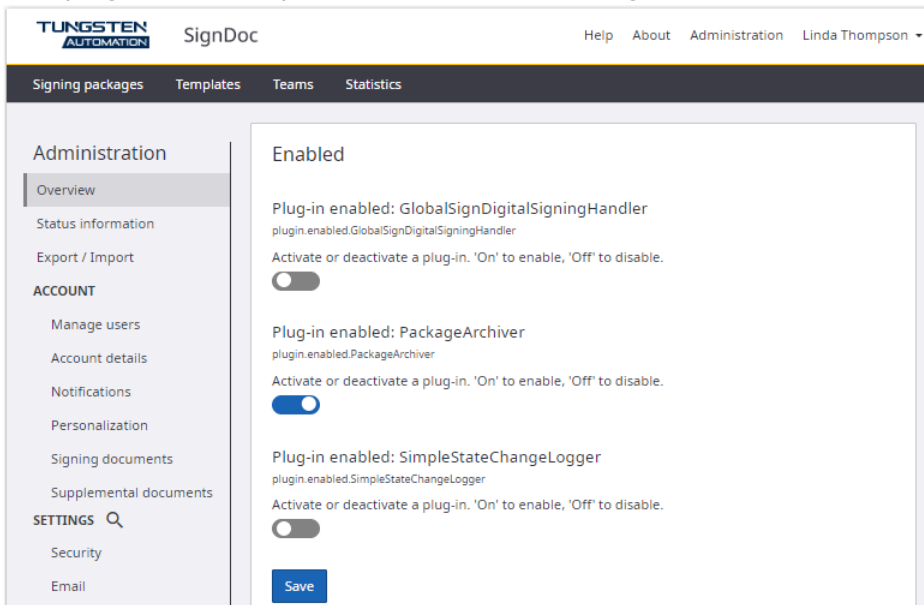
To enable or disable a plug-in in the Administration Center follow these steps.

1. On the **System Settings** menu, click **Plug-ins > Enabled**.
2. Activate or deactivate the plug-in by switching the toggle switch in 'on' (plug-in enabled) or 'off' (plug-in disabled) position and save the settings.



To enable or disable a plug-in in the Manage Client follow these steps.

1. In the Manage Client, on the **Administration** menu, click **Plug-ins > Enabled**.
2. Activate or deactivate the plug-in by switching the toggle switch in 'on' (plug-in enabled) or 'off' (plug-in disabled) position and save the settings.



Assign a plug-in implementation to a specific event

i Before a plug-in can be assigned to an event, it must be loaded.

For the Administration Center follow these steps.

1. On the **System Settings** menu, click **Plug-ins > Plug-in implementations**.
2. Enter the plug-in ID and save the settings.

The screenshot shows the 'Plug-in implementations' page in the SignDoc Administration Center. The page has a dark header with the 'TUNGSTEN AUTOMATION' logo and 'SignDoc' text. Below the header is a navigation bar with 'System settings' highlighted. The main content area is divided into a left sidebar and a main panel. The sidebar contains 'System settings' with a search icon and a list of categories: Security, Email, S/MIME, General, SMTP, Client, Signing, Manage, and Administration. The main panel is titled 'Plug-in implementations' and contains two sections. The first section is 'SignerSearchEvent implementation' with the plugin ID 'plugin.event.impl.signersearchevent'. It defines the ID of the plug-in that handles the SignerSearchEvent event. The default value is 'SignDocSignerSearch'. The second section is 'SigningEvent implementation' with the plugin ID 'plugin.event.impl.signingevent'. It defines the ID of the plug-in that handles the SigningEvent event. The default value is 'SignDocDefaultSigningHandler'. A 'Save' button is located at the bottom of the main panel.

For the Manage Client follow these steps.

1. On the **Administration** menu, click **Plug-ins > Plug-in implementations**.
2. Enter the plug-in ID and save the settings.

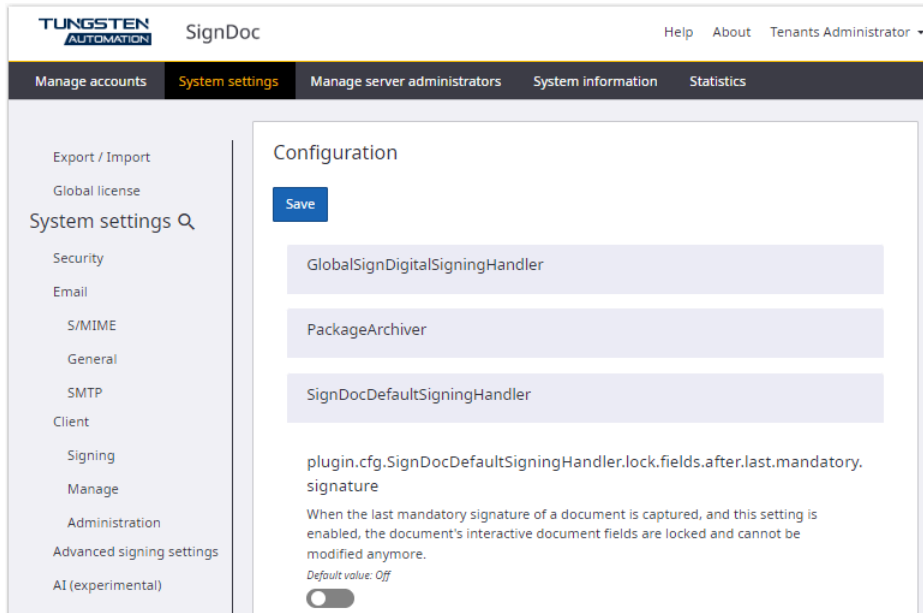
The screenshot shows the 'Plug-in implementations' page in the SignDoc Manage Client. The page has a dark header with the 'TUNGSTEN AUTOMATION' logo and 'SignDoc' text. Below the header is a navigation bar with 'Administration' highlighted. The main content area is divided into a left sidebar and a main panel. The sidebar contains 'Administration' with a search icon and a list of categories: Overview, Status information, Export / Import, ACCOUNT, Manage users, Account details, Notifications, Personalization, Signing documents, and Supplemental documents. The main panel is titled 'Plug-in implementations' and contains two sections. The first section is 'SignerSearchEvent implementation' with the plugin ID 'plugin.event.impl.signersearchevent'. It defines the ID of the plug-in that handles the SignerSearchEvent event. The default value is 'SignDocSignerSearch'. The second section is 'SigningEvent implementation' with the plugin ID 'plugin.event.impl.signingevent'. It defines the ID of the plug-in that handles the SigningEvent event. The default value is 'SignDocDefaultSigningHandler'. A 'Save' button is located at the bottom of the main panel.

After a plug-in has been enabled, it must be assigned to a specific event to modify or extend the SignDoc Standard behavior.

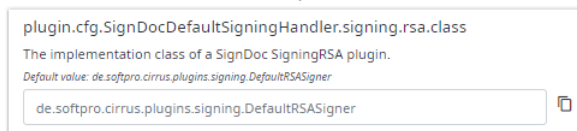
Configure plug-ins

To configure a plug-in in the Administration Center follow these steps.

1. On the **System Settings** menu, click **Plug-ins > Configuration**.
2. Select a plug-in ID from the list.

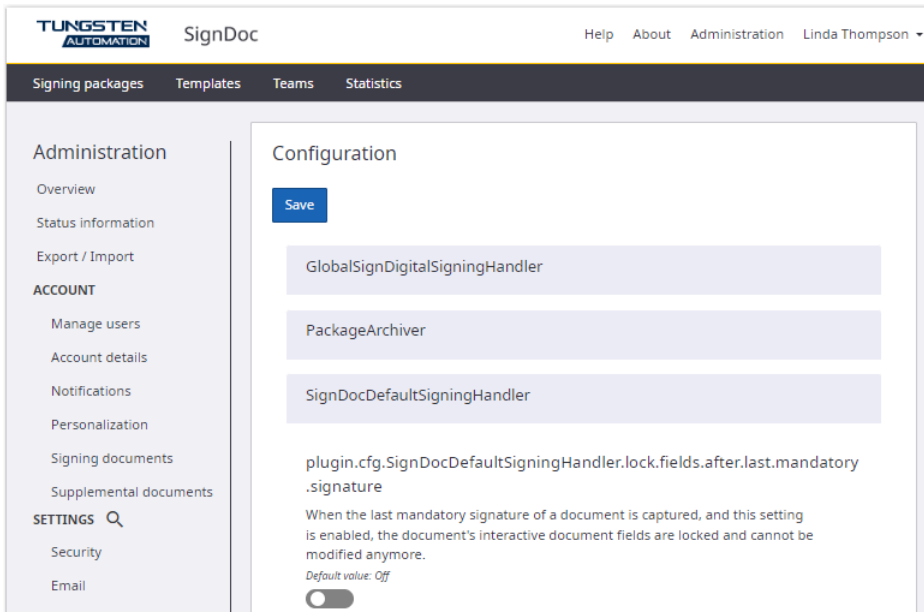


3. Set the value for the implementation class and save the settings.

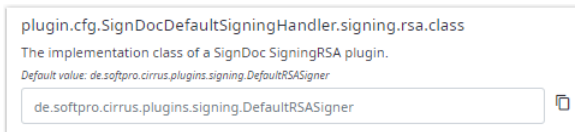


To configure a plug-in in the Manage Client follow these steps.

1. On the **Administration** menu, click **Plug-ins > Configuration**.
2. Select a plug-in ID from the list.



3. Set the value for the implementation class and save the settings.



Plug-ins can provide specific configuration options that can be configured in the Administration Center and in the account administration area of the Manage Client.

Plug-in development

Plug-in interface

SignDoc Standard plug-ins support the 'event' plug-in interface. In addition to that, they support the definition of plug-in configuration data and the parameter description.

A plug-in must implement the following interfaces.

IPlugin

The `de.softpro.sppuginif.IPlugin` interface defines the general plug-in parameters. This includes the:

- Plug-in ID – used to identify the plug-in. This has to be unique.
- General plug-in information – such as vendor, description, copyright.
- Error message information – provide localized information for a particular error.

- Injection point for plug-in configuration information – is called by the application with the plug-in configuration data when the plug-in is instantiated.

IEventPlugin

The `de.softpro.sppluginif.IEventPlugin` interface defines the calling procedure for event based plug-ins. A plug-in defines which events it supports by returning a list with `supportedEvents`. The application will then post an event to `eventCallback` providing a parameter map with event specific content and getting a result map, also with event-specific content.

Each event definition will also provide a list of supported input and output parameters.

IConfigurablePlugin

The `de.softpro.cirrus.plugins.IConfigurablePlugin` interface defines how a SignDoc Standard plug-in can make its configurable parameters known to the application.

The application will use `getSettingDescriptions` to query what configuration settings the plug-in supports. The plug-in will return an array of `PluginSettingDescription`, where each element describes one setting:

- The name used to identify the setting.
- The description of the setting.
- If the setting is mandatory or optional.
- A Java regular expression that can be used to validate the setting value.

This information can be used by the application to provide an administrator GUI to allow plug-in configuration. It is also used by the configuration service to validate entries.

The setting descriptions have to be returned localized, in the locale requested.

Plug-in implementation

It is recommended to use the abstract classes provided where applicable. The `AbstractEventPlugin` can be used as a basis for event plug-ins. If settings are needed, the `IConfigurablePlugin` interface has to be implemented.

For exceptions thrown by the plug-in, use `de.softpro.sppluginif.PluginException` and error numbers defined in `de.softpro.sppluginif.EPluginMsgs` as far as possible. Even though you can define your own exception class and error number range, the application will only be able to react in an individual manner to exceptions it knows about. All custom exceptions will be treated the same as a `PLUGIN_UNKNOWN_EXCEPTION`.

How SignDoc uses plug-ins

- For a plug-in to be used, the server administrator (role SUPERUSER) has to add a plug-in to the load list. After the load list is changed with the configuration service, the application will dynamically reload the plug-ins in the load list.
- For each plug-in on the load list, the application will use the `IConfigurablePlugin` interface, if it is implemented, to query the configuration information that the plug-in supports. For each `PluginSettingDescription` returned, the configuration service will generate a configuration

description according to the values returned. It will also generate an enabled setting for the plug-in ID.

- The server administrator and/or the account administrators can then enable the plug-in globally or for specific accounts by setting the `plugin.enabled.<pluginId>` configuration setting to true. For every enablement an instance of the plug-in will be created. If the plug-in is enabled globally, only one instance will be created that is accessible for all accounts. If the plug-in is enabled on an account-specific basis, each account will use its own instance. This is also true if an account overrides the global setting.
- They can also provide the configuration information according to the settings provided by the plug-in. This information can be account specific.
- The application will inject the configuration information via `injectPluginConfiguration` from the `IPlugin` interface. This is done for every instance of the plug-in on an account-specific basis.
- The plug-in can now be used. The application will post supported events to the plug-in using `eventCallback` from the `IEventPlugin` interface.

Signing plug-in

SigningEvent plug-in description

It is possible to use a SigningEvent plug-in for signing signature fields. This plug-in enables the user to use for example HSM services for signing a signature field or control the appearance of the signature field completely.

For a simple sample implementation, see [Minimal SigningEvent implementation](#).

SignDoc Standard provides a default SigningEvent plug-in. The implementation of this plug-in is described in [SignDocDefaultSigningHandler](#).

Use and configure a custom SigningEvent plug-in

- Deploy the plug-in as described in [Plug-in handling](#).
- Load, enable, assign and configure the plug-in as described in [Plug-in administration](#).

Minimal SigningEvent implementation

SampleSigningHandler is an example for a minimal implementation of a SigningEvent plug-in and can be found in:

```
signdoc_home\interfaces\plugins\cirrus-plugin-samples-*.zip
```

This plug-in uses SignDoc SDK to sign the document's signature field.

SigningRSA interface

This interface is used to delegate the actual calculation of the digital signature. The default implementation calculates the signature in the local SignDoc process using the provided account-specific certificates and private key.

i If it is required to keep the private key in an HSM and calculate the signature at/with a trusted location/entity, it is recommended to use this interface to delegate the signature calculation.

This interface is rather simple and focuses only on the task to calculate the digital signature. For example, a custom implementation does not have to take care for other important required actions, such as set signature appearance or encrypt biometric data, since SignDoc will provide for this using the [SignDocDefaultSigningHandler](#).

For a simple sample implementation, see [Minimal SigningRSA implementation](#).

Configure a custom SigningRSA implementation

1. Make sure that the implementation including dependencies (either single class files or a jar file) is available in the `CIRRUS_HOME/plugins/default` directory. It is not required to restart the SignDoc service after having done this.
 - Single class files must be provided in the `CIRRUS_HOME/plugins/default/classes` directory.
 - Jar files must be provided in the `CIRRUS_HOME/plugins/default/lib` directory.
 - See also [Plug-in handling](#).
2. Log in to the Administration Center or as an account administrator.
3. Assign the full class name of this class, for example `com.mycompany.plugins.MyRSASigner` to the setting:
System Settings > Plug-ins > Configuration > SignDocDefaultSigningHandler > `plugin.cfg.SignDocDefaultSigningHandler.signing.rsa.class`
After applying these settings, the SignDocDefaultSigningHandler will use the assigned class to calculate the RSA signature for signature fields.

Minimal SigningRSA implementation

SampleRSASigner is an example for a minimal implementation of a SigningRSA plug-in and can be found in:

```
signdoc_home\interfaces\plugins\cirrus-plugin-samples-*.zip
```

This plug-in uses the BouncyCastleProvider to calculate the RSA signature based on the provided certificate settings of the SignDoc account.

Core plug-ins

SignDoc default signing handler plug-in

This is the default `SigningEvent` implementation in SignDoc Standard. The `SignDocDefaultSigningHandler` is responsible for the following actions when signing a signature field:

- Signing the PDF Digital signature field
 - By default, the `SignDocDefaultSigningHandler` uses the account-specific signing certificates.
 - This action can be delegated completely to an HSM service if an alternative implementation (see [SigningRSA interface](#)) is provided and configured.
- Assigning the visual appearance of the signed signature field. The appearance depends on the different input methods (handwritten signature, click to sign, sign with image, photo).
- Securely storing biometric data (if applicable) together with the signature field.
 - It uses the account-specific biometric (public) key to encrypt the data.

Configuration

The plug-in is configured in the SignDoc configuration UI in the plug-ins configuration section and provides the following configuration options:

- **plugin.cfg.SignDocDefaultSigningHandler.lock.fields.after.last.mandatory.signature**
When the last mandatory signature of a document is captured, and this setting is enabled, the document's interactive document fields are locked and cannot be modified anymore.
Default value: Off
- **plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters**
[expert topic] Extra parameters for the approval (or certification) signature, that is, the signature proper. The value is an XML document that must conform to `SignDocParameters.dtd`. You can see the available parameters and the possible constants for integer parameters in the documentation of Java class `SignDocSignatureParameters`.
- **plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters.DTS**
[expert topic] Parameters for adding a DTS (Document Time Stamp). If this parameter is not set, no DTS will be added. The value is an XML document that must conform to `SignDocParameters.dtd`. You can see the available parameters and the possible constants for integer parameters in the documentation of Java class `SignDocSignatureParameters`.
- **plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters**
[expert topic] Extra parameters for verification of the approval (or certification) signature, this is, the signature proper. The value is an XML document that must conform to `SignDocParameters.dtd`. You can see the available parameters and the possible constants for integer parameters in the documentation of Java class `SignDocVerificationParameters`.
- **plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DSS**
[expert topic] Parameters for verification of signatures for updating the DSS (Document Security Store). If this parameter is not set, the DSS won't be updated with revocation data. The value is an XML document that must conform to `SignDocParameters.dtd`. You can see the available parameters and the possible constants for integer parameters in the documentation of Java class `SignDocVerificationParameters`.
- **plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DTS**

[expert topic] Extra parameters for verification of the DTS (Document Time Stamp). The value is an XML document that must conform to `SignDocParameters.dtd`. You can see the available parameters and the possible constants for integer parameters in the documentation of Java class `SignDocVerificationParameters`.

- **plugin.cfg.SignDocDefaultSigningHandler.signing.rsa.class**
The implementation class of a SignDoc SigningRSA plug-in.
Default value: `de.softpro.cirrus.plugins.signing.DefaultRSASigner`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.rsa.plugin.configuration**
The configuration of the SignDoc SigningRSA plug-in.
- **plugin.cfg.SignDocDefaultSigningHandler.signing.signmethods.add.signer**
Add signer information to the signature field appearance if the signing method matches one of the listed methods. Valid values: `all,c2s,handwritten,image` (separated by comma). Remark: The value "image" applies to "Sign with image", "Sign with stamp" and "Sign with photo". Example: `handwritten, image`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.signmethods.add.timestamp**
Add a timestamp to the signature field appearance if the signing method matches one of the listed methods. Valid values: `all,c2s,handwritten,image` (separated by comma). Remark: The value "image" applies to "Sign with image", "Sign with stamp" and "Sign with photo". Example: `handwritten, image`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.timestamp.format**
Timestamp format. For valid Syntax, see Java `SimpleDateFormat`. Some Examples (separated by |): `yyyy-MM-dd HH:mm Z | EEE, d MMM yyyy HH:mm:ss Z | MMMM dd, yyyy | MMM/dd/yyyy`. Default value: `yyyy-MM-dd HH:mm Z`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.textline.color**
The color of the text lines. RGB hex-coded value. Examples: `000000 => black, 0000FF => blue`
Default value: `000000`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.textline.position**
The position of text lines. Valid values: `below, overlay`
Default value: `below`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.render.hint.handwritten**
Rendering hints for the code rendering handwritten signatures. Valid values: `anti_alias, gray, bw`
Default value: `anti_alias`
- **plugin.cfg.SignDocDefaultSigningHandler.signing.render.color.handwritten**
HEX color code to render handwritten signatures. A valid HEX color code, such as `#FF0000` for RED. Default: `#000000, Black`
Default value: `#000000`

GlobalSign digital signing handler plug-in

This plug-in uses AATL certificates provided by the GlobalSign web service to sign a signature field and the final document.

Prerequisites

To complete the signing process using GlobalSign provided AATL certificates, a user should have correct rights and credentials to access the GlobalSign REST API.

The mandatory prerequisites to use GlobalSign are:

1. The user should create an identity and it should be active.
2. The user should create a Digital Signing Service (DSS) which provides fully trusted Adobe AATL digital signatures and timestamps.
3. The user should create API credentials and link them to an active DSS service. The API credentials created here have two parts API key and API secret which is essential to access the DSS API.
4. Create mTLS certificate and link the API credentials to this mTLS certificate, needed for SSL handshakes.

GlobalSign DSS is integrated into SignDoc by using a new `IConfigurablePlugin` Signing Handler named as `GlobalSignDigitalSigningHandler`. Along with `SignRSA` implementation 'GlobalSignRSA' this handler is loaded into the `plugin.loadlist` by default. The user should enable it and update the `plugin.event.impl.signingevent` to `GlobalSignDigitalSigningHandler` to use it for signing.

The user can then set the mandatory `GlobalSignDigitalSigningHandler` plug-in configurations. The below plug-in configurations are mandatory and if not set the signing will result in error.

- **`plugin.cfg.GlobalSignDigitalSigningHandler.api.key`**
GlobalSign DSS API credentials API key.
- **`plugin.cfg.GlobalSignDigitalSigningHandler.api.secret`**
GlobalSign DSS API credentials API secret.
- **`plugin.cfg.GlobalSignDigitalSigningHandler.api.url`**
GlobalSign DSS API URL.
- **`plugin.cfg.GlobalSignDigitalSigningHandler.mtls.certificate`**
GlobalSign API mTLS certificate, issued by GlobalSign mTLS CA.
- **`plugin.cfg.GlobalSignDigitalSigningHandler.mtls.certificate.password`**
GlobalSign API mTLS certificate password.

The above configuration settings are checked for validity and the results are shown in the **Status Information** view in the administration section. In case of errors, the relevant error information is displayed so that the user can make the necessary changes to successfully use GlobalSign. However, this error does not restrict the user to create and start a signing package. In such scenarios the signer will see the error message upon signing and will not be able to complete the signing ceremony.

The user can also set the below optional configuration settings for signature text lines.

- **`plugin.cfg.GlobalSignDigitalSigningHandler.signing.signmethods.add.signer`**
Add signer information to the signature field appearance if the signing method matches one of the listed methods. Valid values: all,c2s,handwritten,image (separated by comma). Remark: The value "image" applies to "Sign with image", "Sign with stamp" and "Sign with photo". Example: handwritten, image.
- **`plugin.cfg.GlobalSignDigitalSigningHandler.signing.signmethods.add.timestamp`**
Add a timestamp to the signature field appearance if the signing method matches one of the listed methods. Valid values: all,c2s,handwritten,image (separated by comma). Remark: The

value "image" applies to "Sign with image", "Sign with stamp" and "Sign with photo". Example: handwritten,image.

- **plugin.cfg.GlobalSignDigitalSigningHandler.signing.timestamp.format**
Timestamp format. For valid Syntax, see Java SimpleDateFormat. Some Examples (separated by |): yyyy-MM-dd HH:mm Z | EEE, d MMM yyyy HH:mm:ss Z | MMMM dd, yyyy | MMM/dd/yyyy.
Default value: yyyy-MM-dd HH:mm Z
- **plugin.cfg.GlobalSignDigitalSigningHandler.signing.textline.color**
The color of the text lines. RGB hex-coded value. Examples: 000000 => black, 0000FF => blue.
Default value: 000000
- **plugin.cfg.GlobalSignDigitalSigningHandler.signing.textline.position**
The position of text lines. Valid values: below, overlay.
Default value: below
- **plugin.cfg.GlobalSignDigitalSigningHandler.pades.profile.conformance.level**
PAdES Baseline Profile Conformance Level as defined by ETSI - TS 103 172. Valid levels: PAdES_B-B, PAdES_B-BT, PAdES_B-LT, PAdES_B-LTA. Defines the minimum conformance level. The signature might have a higher level than defined, depending on the state of the original document.
Default value: PAdES_B-LT
- **plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.certificate.ou**
The name of the department or organization unit making the request. Length must be less or equal to 32 characters. This value will be used if the GlobalSign DSS certificate has a non STATIC requirement for the COMMON NAME (CN).
- **plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.certificate.document.cn**
A meaningful name for the certificate's Common Name (CN) attribute that is used for digital signatures that are not originating from a signer (such as your company name for sealing documents). For signers, the Common Name will be automatically set to the signer's name. Length must be less or equal to 64 characters. This value will be used if the GlobalSign DSS certificate has a non STATIC requirement for the COMMON NAME (CN).
Default value: Tungsten SignDoc
- **plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.retries.on.ratelimit**
Defines the maximum number of retries attempts the plug-in should do if a GlobalSign request returned status code 429 (Too Many Requests).
Default value: 3
- **plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.retries.wait.time**
Defines how long the plug-in should wait in seconds before retrying a failed GlobalSign API request.
Default value: 5
- **plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.cache.identity.time**
Defines how long the server will cache an authenticated GlobalSign identity in minutes.
Default value: 9
- **plugin.cfg.GlobalSignDigitalSigningHandler.signing.render.color.handwritten**
HEX color code to render handwritten signatures. A valid HEX color code, such as #FF0000 for RED. Default: #000000, Black.
Default value: #000000
- **plugin.cfg.GlobalSignDigitalSigningHandler.signing.render.hint.handwritten**

Rendering hints for the code rendering handwritten signatures. Valid values: `anti_alias`, `gray`, `bw`.
Default value: `anti_alias`

The final PDF document created for the signing package consisting of all the documents of the package along with the audit trails and the package information will also be signed using the `GlobalSignDigitalSigningHandler` signing event only if a signing certificate is not uploaded at the account level for signing the documents. If a signing certificate is present at the account level, the final document will be signed using this signing certificate, however the signature fields will be signed using the `GlobalSignDigitalSigningHandler`.

Run GlobalSign for the DSS test certificate

GlobalSign production root certificate does not support GlobalSign DSS test services. Hence for the GlobalSign test DSS user should add `de.softpro.cirrus.plugins.signing.GlobalSignDigitalSigningHandlerTest` in the `plugin.loadlist`.

`GlobalSignDigitalSigningHandlerTest` supports all configurations in `GlobalSignDigitalSigningHandler` except:

- `plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.root.certificate`
- `plugin.cfg.GlobalSignDigitalSigningHandler.globalsign.intermediate.certificate`
- `plugin.cfg.GlobalSignDigitalSigningHandler.pades.profile.conformance.level`

Also, the signature signed using this option will not be trusted by Adobe Reader.

Notification plug-in

Notification plug-in description

The notification plug-in interface is used to access a notification service to send out notification data. Currently this is the 2-factor authentication code a signer has to enter when an authentication code is required.

The sample implementation uses an SMS service (Clickatell) to send out the notification information via SMS.

Supported events

The notification service supports two events:

- **Notification parameter event** - used to query the parameter information available.
- **Notification event** - used to actually send a notification.

The application will usually first issue a parameter event to check what parameters a particular notification channel needs (in case of a SMS notification for instance the phone number). It will generate input fields for the parameters described by the plug-in to capture the necessary parameters.

Once a signer needs to be notified, the application will issue a notification event, providing the parameters captured above.

Notification parameter event

The `de.softpro.cirrus.plugins.event.NotificationParametersEvent` describes the notification parameters event. This event is used to query the parameters of a particular notification service.

The event input parameters describe what is requested:

- **IN_INFORMATION_LIST**

The list of information to be queried. Can be:

`IN_INFO_TYPE_DESCRIPTION` (return the description of the notification service ())

`IN_INFO_MAX_MESSAGE_SIZE` (return the maximum message size that can be sent out in one message based on the current plug-in configuration)

`IN_INFO_PARAMETERS` (return the parameter descriptions needed to use the notification event)

If no selection list is supplied, all of the above will be returned.

- **IN_LOCALE**

The locale the information should be returned in (IETF BCP 47 tag). If not specified the default locale is English.

The event will return the requested information as a list of output parameters containing `OUT_NOTIFICATION_TYPE_DESCRIPTION`, `OUT_MAX_MESSAGE_SIZE` and `OUT_PARAMETERS`. In case of `OUT_PARAMETERS` the parameter will contain a list of `NotificationTargetParameterDescription` objects, that each describe one input parameter to the notification event:

- name
- help text
- placeholder
- validation regular expression (used to validate the input)

The information above should be returned in the language specified by the `IN_LOCALE` parameter.

Notification event

The `de.softpro.cirrus.plugins.event.NotificationEvent` describes the event issued to actually trigger a notification.

The event input parameters are:

- **IN_TARGET**

A map of target parameters described by the list retrieved by the previous event.

- **IN_MESSAGE**

The message to be sent.

There are no output parameters.

Core plug-ins

SMS notification plug-in

- [Notification and the SMS plug-in](#)
- [Registering an account with the SMS service](#)
- [Configuration](#)

Notification and the SMS plug-in

The notification service is designed to send a message to the user / signer. For a 2-factor authentication the login information is split into two parts: the login link and the access code.

The access code has to be delivered via a different channel than the login link.

To be able to support different types of delivery channels, the feature is implemented via a plug-in. Thus, additional delivery channels can be supported without changes to the product.

The core plug-in supports an SMS notification channel.

Registering an account with the SMS service

To use the SMS plug-in, you will have to own a user account with Clickatell. See the Clickatell website.

Clickatell changed its account structure and API in November 2016. Therefore two different plug-ins are provided, depending on the type of account you have registered:

- Accounts registered before November 2016 (Clickatell Central):
Use the `NotificationSMSClickatell` plug-in.
- Accounts registered after November 2016 (Clickatell Platform):
Use the `NotificationSMSClickatellPlatform` plug-in.

The plug-ins can be used at the same time in an installation. Usually one account will only use a single plug-in. Both plug-ins return 'SMS' as a delivery channel.

Configuration

All settings described here are configured via the Cirrus configuration service.

Currently the configuration service is reachable via the REST API, or the configuration editors in the Manage Client or Administration Center.

The following sections describes the configuration for each type of plug-in.

Clickatell Central API

Plug-in load list

The load list specifies which plug-ins are supported by Cirrus. To make the SMS notification plug-in usable, it has to be added to the load list.

The load list is a ',' delimited list of plug-in class names.

```
plugin.loadlist =  
de.softpro.cirrus.plugins.notification.NotificationSMSClickatell
```


Enabling the plug-in

Once the plug-in has been loaded via the load list it can be enabled:

- For one or more specific accounts
- For all accounts globally

To enable the plug-in, you have to set the setting

```
plugin.enabled.NotificationSMSClickatell = true
```

either as a global setting (no account) or for a specific account ID.

SMS plug-in settings

Plug-in settings are set as:

```
plugin.cfg.<pluginId>.<settingName>
```

Example

```
plugin.cfg.NotificationSMSClickatell.url
```

Following settings are supported by the `NotificationSMSClickatell` plug-in:

- **url** SMS service URL. The URL where the Clickatell SMS service can be reached. Usually `https://api.clickatell.com/http/sendmsg`
- **userid** SMS service user ID. The user ID of the Clickatell SMS service user used to authenticate with the service.
- **password** SMS service password. The password of the Clickatell SMS service user used to authenticate with the service.
- **apiid** SMS service API ID. The API ID you received when setting up the user and http service at Clickatell.
- **senderid** Sender ID to use (numeric only 16 digits or alphanumeric 11 characters, registered). If you want to use a sender ID with the sent SMS messages, you have to register the sender ID at Clickatell. After successful registration you can specify the registered sender ID here. The sender ID can be either a telephone number, or an alphanumeric ID. Alphanumeric sender IDs are limited to 11 characters in length.
- **utf16** Use UTF-16BE encoding (true / false). The SMS alphabet is limited regarding the characters that can be used for the message. If special characters or locales (Chinese, Japanese, and more) or symbols will be used in the message, UTF16 encoding can be used. The UTF encoding however limits the message length. Thus it should only be used if necessary.
- **maxparts** Maximum number of message parts to be used (1-3, default 3). SMS messages are limited in length. Longer messages can be sent by chaining SMS parts together (at additional cost). This setting specifies the maximum number of message parts the system will send.
- **userparam** Additional parameters to be sent to the service (use URL encoding). If additional Clickatell settings need to be used they can be specified here. The parameters will have to be URL encoded.

Testing howto

The steps needed with the Swagger UI to get a plug-in configured are described below:

1. Get an access token for `ksdadmin` (no account).

Use Users > Create an authentication token.

2. Set the plug-in load list account independent.
Use Configuration > Set configuration settings.

```
[
  {
    "k": "plugin.loadlist",
    "v": "de.softpro.cirrus.plugins.notification.NotificationSMSClickatell"
  }
]
```

3. Set the plug-in cfg (can be account specific).
Use Configuration > Set configuration settings.

```
[
  {
    "k": "plugin.enabled.NotificationSMSClickatell",
    "v": "true"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatell.url",
    "v": "http://smscatterer.sdlabs.de:8080/sendmsg"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatell.userid",
    "v": "test_user"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatell.password",
    "v": "test_password"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatell.apiid",
    "v": "1234567"
  }
]
```

Clickatell Platform API

Plug-in load list

The load list specifies which plug-ins are supported by Cirrus. To make the SMS notification plug-in usable, it has to be added to the load list.

The load list is a ',' delimited list of plug-in class names.

```
plugin.loadlist =
de.softpro.cirrus.plugins.notification.NotificationSMSClickatellPlatform
```

Enabling the plug-in

Once the plug-in has been loaded via the load list it can be enabled:

- For one or more specific accounts
- For all accounts globally

To enable the plug-in, you have to set the setting

```
plugin.enabled.NotificationSMSClickatellPlatform = true
```

either as a global setting (no account) or for a specific account ID.

SMS plug-in settings

Plug-in settings are set as:

```
plugin.cfg.<pluginId>.<settingName>
```

Example

```
plugin.cfg.NotificationSMSClickatellPlatform.url
```

Following settings are supported by the `NotificationSMSClickatellPlatform` plug-in:

- **url** SMS service URL The URL where the Clickatell SMS service can be reached. Usually `https://platform.clickatell.com/messages/http/send`
- **apikey** SMS service API key. The Clickatell Platform API key you have set up by creating a new http integration on your Clickatell account.
- **senderid** Optional sender ID to use (numeric only 16 digits or alphanumeric 11 characters, registered). If you want to use a sender ID with the sent SMS messages, you have to register the sender ID at Clickatell. After successful registration you can specify the registered sender ID here. The sender ID can be either a telephone number, or an alphanumeric ID. Alphanumeric sender IDs are limited to 11 characters in length.

Testing howto

The steps needed with the Swagger UI to get a plug-in configured are described below:

1. Get access token for ksdadmin (no account).
Use Users > Authentication.
2. Set plug-in load list account independent.
Use Configuration > Set configuration settings.

```
[
  {
    "k": "plugin.loadlist",
    "v":
      "de.softpro.cirrus.plugins.notification.NotificationSMSClickatellPlatform"
  }
]
```

3. Set plug-in cfg (can be account specific).
Use Configuration > Set configuration settings.

```
[
  {
    "k": "plugin.enabled.NotificationSMSClickatellPlatform",
    "v": "true"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatellPlatform.url",
    "v": "https://platform.clickatell.com/messages/http/send"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatellPlatform.apikey",
    "v": "Your-API-key=="
  },
]
```

Package state change plug-in

Package state change plug-in description

The state change event lets you write a plug-in that can perform additional processing whenever a signing package or a signer change state.

The plug-in should publish which state changes it wants to process, to avoid generating too many audit trail entries for state changes where no action is taken. See the `SupportedStateChange` event below.

State change events will be sent to all enabled plug-ins that support the state change events.

Supported events

The following events are supported:

- Supported state change event
- Package state change event
- Signer state change event

Supported state change event

The `SupportedStateChange` event is generated for each state change (both signing package and signer state changes) before the main event is triggered. It is used to inquire if the plug-in intends to process the state change specified.

If the plug-in responds with `true`, the main state change event is triggered and the relevant audit trail entries are generated. If the plug-in responds `false`, the main state change event will not be sent to the plug-in.

If the plug-in does not implement this event, the main state change event will be sent to it.

If additional information is needed by the plug-in, it can use a REST API call to query package or signer information. For this purpose, the owner token parameter is provided, that can be used to authenticate the REST API call.

The input event parameters describe what state change will be processed:

- **IN_EVENT_TYPE**
The type of the state change. Can be either `TYPE_SIGNINGPACKAGE` or `TYPE_SIGNER`.
- **IN_OLD_STATE**
The state (string) before the state change.
- **IN_NEW_STATE**
The new state (string) after the state change.

The output parameters give the plug-in response as to if it intends to process that particular change or not:

- **OUT_SUPPORTED**

The plug-in answer if it intends to process the state change given by the input parameters. The type is Boolean. If the response is TRUE, the event will be passed to the plug-in. If FALSE, the event will not be passed and no audit trail entries will be generated.

Package state change event

The `PackageStateChange` event will be generated when a signing package change state. This does not apply for the initial signing package state upon creation.

The input event parameters give the details of the signing package state change:

- **SIGNDOC_HOME (SIGNDOC_HOME)**
The location of the SIGNDOC_HOME or CIRRUS_HOME directory.
- **IN_PACKAGE_DATA (packageData)**
A `StateChangeEventPackage` object that provides useful information of the current signing package.
- **IN_ACCOUNT_OID (accountOid)**
The account OID (string) of the signing package account.
- **IN_SIGNINGPACKAGE_OID (signingpackageOid)**
The signing package OID (string).
- **IN_SIGNINGPACKAGE_NAME (signingpackageName)**
The signing package name (string).
- **IN_OLD_STATE (oldState)**
The signing package state before the state change (string).
- **IN_NEW_STATE (newState)**
The new signing package state after the state change (string).
- **IN_OWNER_TOKEN (ownerToken)**
The authentication token that can be used to authenticate a REST API call to obtain further package details.

There are no output parameters.

Signer state change event

The `SignerStateChange` event will be generated when a signer changes state. This does not apply for the initial signer state upon creation.

The input event parameters give the detail of the signer state change:

- **SIGNDOC_HOME (SIGNDOC_HOME)**
The location of the SIGNDOC_HOME or CIRRUS_HOME directory.
- **IN_ACCOUNT_OID (accountOid)**
The account OID (string) of the signing package account.
- **IN_SIGNINGPACKAGE_OID (signingpackageOid)**
The signing package OID (string).
- **IN_SIGNER_OID (signerOid)**
The signer OID (string).
- **IN_SIGNER_FIRST_NAME (signerFirstName)**
The signer first name, if recorded (string, optional).

- **IN_SIGNER_LAST_NAME (signerLastName)**
The signer last name, if recorded (string, optional).
- **IN_SIGNER_DISPLAY_NAME (signerDisplayName)**
The signer display name (usually first name + last name), if recorded (string, optional).
- **IN_OLD_STATE (oldState)**
The signer state before the state change (string).
- **IN_NEW_STATE (newState)**
The new signer state after the state change (string).
- **IN_OWNER_TOKEN (ownerToken)**
The authentication token that can be used to authenticate a REST API call to obtain further package and signer details.

There are no output parameters.

Core plug-ins

KTA state change plug-in

- [KTA](#)
- [State change events](#)
- [Configuration](#)

KTA

This plug-in implements the communication with the Tungsten TotalAgility system. The plugin will create signing packages with documents to be signed as part of its workflow. After the documents have been signed, this plug-in will report the status change back to the Tungsten TotalAgility system and let it continue its workflow.

Refer to the Tungsten TotalAgility documentation on how to configure the Tungsten TotalAgility system to generate signing packages with SignDoc.

State change events

The plug-in implements the state change event interface and reacts to SignDoc state changes:

- Signing package state changes: Change from any state to CANCELED, REJECTED, EXPIRED or COMPLETE
- Signer state changes: Change from any state to COMPLETE

Upon receiving the relevant state change events, the plug-in will generate KTA calls to inform KTA on the new processing state.

Configuration

Plug-in load list

The load list specifies which plug-ins are supported by Cirrus. To make the KTA state change plug-in usable, it has to be added to the load list.

The load list is a ',' delimited list of plug-in class names.

```
plugin.loadlist =  
de.softpro.cirrus.plugins.state_change.KTAStateChangeNotification
```

Enabling the plug-in

Once the plug-in has been loaded via the load list it can be enabled:

- For one or more specific accounts individually
- For all accounts globally

To enable the plug-in, you have to set the setting

```
plugin.enabled.KTAStateChangeNotification = true
```

either globally (no account ID), or for a specific account.

Plug-in settings

Plug-in settings are set as

```
plugin.cfg.<pluginId>.<settingName>
```

Example

```
plugin.cfg.KTAStateChangeNotification.ktaurl
```

Following settings are supported by the KTA state change notification plug-in:

- **ktaurl** The KTA URL to send information to.
- **cirrusurl** The Cirrus (SignDoc Standard) REST API URL. The plug-in will use the REST API to obtain additional signing package and signer information needed to process the request. Specify the URL without the REST API version number! Example: `http://your.host.name/cirrus/rest`
- **sessionid** The KTA session ID.
- **jobnotetemplate** Optional. If you need to change the job note template you can set this parameter. This was previously done by providing a job note template in the Cirrus home directory.

Additional required configuration setting

If the KTA state change plug-in is enabled it is required that the setting

```
cirrus.document.prepare.msword.signatureline.signerid.source = field_name
```

in the category "Documents and packages" is set.

Package archiver plug-in

This plug-in implements the `StateChangeEvent`. The plug-in stores signing package information in separate files depending on their current workflow state. These files can be collected and processed by third-party systems.

Each file contains the PDF documents of the signing package along with metadata. Custom metadata can be stored in the signing package with the custom attribute of the signing package that can be set with the REST API.


Usage

Before the plug-in can be used, it must be enabled in the plug-ins configuration section. This can be done system-wide in the Administration Center or per account in the Administration section by an account administrator.

The `plugin.cfg.PackageArchiver.package.states` configuration option defines for what state changes of the signing package the plug-in will write a file to the local file system. By default only the COMPLETE state is considered whenever a signing package is completed. It is possible to concatenate more states with comma, to archive the signing package for different states.

The location of the files is defined by:

```
plugin.cfg.PackageArchiver.archive.folder
```

 UNC paths are not supported.

The file name and directory structure can be defined with:

```
plugin.cfg.PackageArchiver.timestamp.format
```

```
plugin.cfg.PackageArchiver.filename.pattern.
```

The output format is defined with the boolean switches:

```
plugin.cfg.PackageArchiver.archive.export.xml
```

```
plugin.cfg.PackageArchiver.archive.export.json
```

Source code

The source code of the plug-in is available as `PackageArchiverSample.java` and can be used as a starting point for own implementations.

```
signdoc_home/interfaces/plugins/cirrus-plugin-samples-<version>.zip
```

Configuration

The plug-in is configured in the SignDoc configuration UI in the plug-ins configuration section and provides the following configuration options:

- **plugin.cfg.PackageArchiver.package.states**
The package states to consider as comma separated list. Valid states: PREPARED, STARTED, COMPLETE, REJECTED, EXPIRED, CANCELED, ARCHIVED
Default value: COMPLETE
- **plugin.cfg.PackageArchiver.archive.folder**
The folder to archive the files. A forward or backward slash can be used to separate directories.
Default value: SIGNDOC_HOME/archive/PackageArchiver
- **plugin.cfg.PackageArchiver.timestamp.format**

The timestamp format. See JavaDoc of SimpleDateFormat for more syntax and examples.

Default value: yyyy-MM-dd_HH-mm-ss

- **plugin.cfg.PackageArchiver.filename.pattern**
Defines how the file name is constructed. A forward or backward slash defines a subdirectory. Possible placeholders are: TIMESTAMP, ACCOUNTID, PACKAGEID, PACKAGESTATE
Default value: ACCOUNTID/PACKAGESTATE/TIMESTAMP_PACKAGEID
- **plugin.cfg.PackageArchiver.archive.export.xml**
Export package data in XML format
Default value: On
- **plugin.cfg.PackageArchiver.archive.export.json**
Export package data in JSON format
Default value: Off

Signer search plug-in

Signer search plug-in description

The signer search plug-in enables you to write a plug-in that performs a signer search which is used by the Manage Client for the signer entry "autocomplete" function in the package wizard. This functionality should help the user to find a specific signer which should sign a signing package.

After entering some characters in the name field of the "Add recipients" section of the "Recipients & Documents" view of the package wizard a signer name search is triggered by the client in order to get a list of signers which contain the entered string as part of their name. The same functionality is provided for the email address of the wanted signer.

The screenshot shows the 'Add recipients' section of a software interface. At the top, there is a toggle switch labeled 'Complete in any order' which is turned on. Below this, there are two input fields: 'Signer' with a dropdown arrow and 'Email'. The 'Signer' field contains the text 'person'. To the right of the 'Email' field are four icons: a plus sign, a minus sign, a gear, and a trash can. Below the input fields, there is a list of search results. The first result is 'Important Person' with the email address '(macgayver@important.com)'. The second result is 'Not so important person' with the email address '(charliebrown@peanuts.com)'. To the left of the search results, there is a button labeled 'Add Recipient' with a plus icon. Below the search results, there is a button labeled 'Add document'.

If one or more signers could be found according the provided name and/or email address part a list of name and email pairs are returned to the client.

The following REST API is used for retrieving the signer list:

```
GET /rest/v8/signerlist
```

The query parameters are:

- **searchname**

The signer name for searching. A signer is included only if his name contains the provided text. This is case-insensitive. The parameters `searchname` and `searchemail` can be set at the same time.

- **searchemail**

The signer email for searching. A signer is included only if his email address contains the provided text. This is case-insensitive. The parameters `searchname` and `searchemail` can be set at the same time.

- **limit**

The number of results to be returned. If 0 or no limit is set then the value from configuration entry `cirrus.rest.resultset.size.max.signers.autocomplete` is used as default limit.

- **custom**

Any custom character data which is passed through to the called plug-in.

The REST API implementation gets the signer list by calling the plug-in which supports the `SignerSearchEvent`.

The default `SignDocSignerSearch` implementation performs a search on the SignDoc database within the current account of the requesting user.

Since only one plug-in implementation of `SignerSearchEvent` can be used for a signer search, it is necessary to define a specific plug-in ID for this event which is the SignDoc internal `SignDocSignerSearch` by default. Prerequisite for the usage of the plug-in is that the specified plug-in definition is included in the `plugin.loadlist` and it must be enabled, either globally or account specific.

Which plug-in implementation is used for the "autocomplete" signer search is defined in the account specific setting `plugin.singleton.id.signersearchevent` which contains the unique ID of `SignerSearchEvent` plug-in. The account independent default setting is `SignDocSignerSearch` for this configuration entry.

If no plug-in is enabled for the `SignerSearchEvent` event then the "autocomplete" function is suspended because the REST call would produce always an empty list.

Supported events

The following event is supported:

- Signer search event

Signer search event

The following input parameters are provided in the `SignerSearchEvent`:

- **IN_USER_ID**

The plug-in can use the requesting `userid` and user's email for identification.

- **IN_USER_EMAIL**

The email address of the requesting user.

- **IN_SIGNER_NAME**

Substring of the requested signer name. The plug-in must support the search for name and/or email based on a substring of the search criteria.

- **IN_SIGNER_EMAIL**

Substring of the requested signer email address. Can be provided as combination together with `IN_SIGNER_NAME`.

- **IN_LIMIT**

The limit input parameter contains the maximum number of result entries which are returned to the caller.

- **IN_CUSTOM**

Additional custom input which can be provided via REST interface. This optional input attribute is not considered in the default plug-in implementation `SignDocSignerSearch`.

The following output is expected from the plug-in for this event:

- **OUT_SIGNER_LIST**

The output contains a list of `SignerSearchResult` (class) elements with the attributes name and email. The list is empty if the search has no result.

The default plug-in implementation `SignDocSignerSearch` performs a substring search on all signers in all packages within the account where the requesting user belongs to.

The search criteria are either the substring of the signer name or a substring of the signer's email or a combination of both if name and email are provided.

The search is distinct without duplicate entries.

Core plug-ins

Document scan plug-in description

The document scan event lets you write a plug-in that can verify the content of an uploaded document or supplemental document. This can be used to verify or validate the content. The most common use is to scan uploaded documents for viruses.

A sample implementation using the open source ClamAV scanner is available, but the customer can implement an interface to the scanner of his choice.

Supported events

The following event is supported

- Document scan event

Document scan event

The `de.softpro.cirrus.plugins.event.document_scan.DocumentScanEvent` describes the document scan event. Whenever a document or a supplemental document is uploaded, a document scan event is posted to all plug-ins registering this event and enabled for the account. If any of the plug-ins returns an invalid result, the document will be rejected.

The event input parameters are:

- **IN_CONTENT**

The document content (binary, byte array), required

- **IN_NAME**

The name of the file being uploaded (string), optional

The output parameters are:

- **OUT_RESULT**

The Boolean result of the scanning, required. True means a scanning issue has been detected, false that the document does not contain any issues (viruses).

- **OUT_CAUSE**

The cause of scanning problems found (string), optional. Can be one of:

OUT_CAUSE_GENERAL – no specific cause information, default

OUT_CAUSE_VIRUS – a virus has been detected

OUT_CAUSE_INVALID_TYPE – an invalid document type has been uploaded

OUT_CAUSE_CONTENT – the document contains invalid content

- **OUT_DETAILS**

Details on the failure (string), optional. If specified, the information will be logged. Can provide additional information on the cause of failure, like the type of virus that has been detected.

Core plug-ins

ClamAV virus scan plug-in

- [Document scan event](#)
- [ClamAV virus scanner](#)
- [Configure the ClamAV document scan plug-in](#)
- [Test the scanning](#)

Document scan event

To prevent invalid documents from being uploaded or viruses being spread to customers via uploaded infected documents, SignDoc Standard supports the document scan event interface.

Whenever a document or supplemental document is being uploaded, a document scan event is posted to all registered plug-ins on the affected account. All plug-ins can scan the document content. If any plug-in responds with 'true' as to invalid content being detected, the document upload is rejected.

A sample implementation of this plug-in for the ClamAV virus scanner is provided. The customer can implement any other scan implementation of his choice, if a different scanner is required.

ClamAV virus scanner

ClamAV

For the sample document scan implementation the ClamAV virus scanner has been used. The main reasons for this choice are that ClamAV:

- Is an open source virus scanner.
- Is available free of charge.

- Is available on all major operating systems.

For detailed information on ClamAV, see the ClamAV website.

Tungsten Automation does not provide support on installing or running a ClamAV server.

Running the ClamAV scan server

To scan documents via the sample ClamAV document scan plug-in, a ClamAV server and a ClamAV REST endpoint need to be running.

- For documentation on running a ClamAV server, see the ClamAV website.
- The REST endpoint is documented under <https://github.com/solita/clamav-rest>

The easiest way to run this combination is using a Docker Compose setup (docker-compose.yml):

```
version: '3'
services:
  clamav:
    image: mkodockx/docker-clamav
    ports:
      - "3310:3310"
  clamav-rest:
    image: lokori/clamav-rest
    links:
      - clamav
    ports:
      - "8080:8080"
    environment:
      - CLAMD_HOST=clamav
```

The above docker-compose.yml shows how to start two Docker containers:

- clamav runs a normal ClamAV server
- clamav-rest runs the REST endpoint connected to clamav server

The compose setup can be started using 'docker-compose up -d'. The URL endpoint for this setup will be http://hostname:8080/scan.

Tungsten Automation does not provide support on installing or running the ClamAV server. Refer to the links above and to the Docker website for additional information.

Configure the ClamAV document scan plug-in

Plug-in load list

The load list specifies which plug-ins are supported by Cirrus. To make the ClamAV document scan plug-in usable, it has to be added to the load list.

The load list is a ',' delimited list of plug-in class names.

```
plugin.loadlist = de.softpro.cirrus.plugins.document_scan.ScanClamAV
```

Enabling the plug-in

Once the plug-in has been loaded it can be enabled:

- For one or more specific accounts individually

- For all accounts globally

To enable the plug-in you have to change the setting

```
plugin.enabled.ScanClamAV = true
```

either globally (no account ID), or for a specific account.

Plug-in settings

Plug-in settings are set as

```
PLUGIN.CFG.<PLUGINID>.<SETTINGNAME>
```

Example

```
plugin.cfg.ScanClamAV.url
```

Following settings are supported by the ScanClamAV plug-in:

- **url** The URL to the ClamAV server REST endpoint (see above). For the sample server configuration provided it has the form:
`http://hostname:8080/scan`

Test the scanning

To test virus detection you can upload the test EICAR virus signature as a document and verify that the scanner works. See the Eicar website.

TSP plug-in

Trusted service provider plug-in description

The trusted service provider interface is used to add a TSP digital signature to the document if a signer is registered with the TSP.

In SignDoc Standard the `IConfigurablePlugin` interface and the account-specific instantiation are used to inject account-specific configuration data via `IPlugin`.

Supported events

The trusted service provider interface supports the following events:

- TSP info event
Provides information about the TSP provider the plug-in implements.
- TSP validation event
Used to validate the credentials of a signer for a specific type of digital signature, before the actual signing takes place.
- TSP signing event
Used to actually sign a document via the trusted service provider.

In case a specific provider does not support the validation step, the validation event does not need to be supported.

TSP info event

The `de.softpro.cirrus.plugins.event.tsp.TSPInfoEvent` returns information specific to the TSP provider this plug-in supports:

- **IN_LOCALE**

Optional. The locale information should be returned in (IETF BCP 47 tag). If not specified, the default locale is English.

- **IN_SETTINGS**

Optional. If present, the settings map will completely override the settings provided at plug-in instantiation via `IPlugin.injectPluginConfiguration`.

The output will provide provider-specific information needed to display input and information pages related to the TSP validation and signing process:

- **OUT_CREDENTIALS_DESCRIPTION**

The list of validation credentials needed by the TSP to perform a validation and/or signing. See below.

- **OUT_PROVIDER_INFO**

Provider-specific information that will be used when displaying input pages (descriptions, help, registration URL). This returns a map of settings described below.

- **OUT_PROVIDER_NAME**

The (localized) name of the trusted service provider implemented by this plug-in.

Currently supported provider info fields are:

- **PI_VALIDATION_TEXT** ("validation_text"): An optional text that describes the provider-specific validation procedure.
- **PI_SIGNING_TEXT** ("signing_text"): An optional text that describes the provider-specific signing procedure.
- **PI_HELP_TEXT** ("help_text"): An optional text that provides help and background information regarding the TSP provider.
- **PI_REGISTRATION_URL** ("registration_url"): An optional URL to the provider registration page where a signer can register a new user with this provider.

Each validation credential description element

(`de.softpro.cirrus.plugins.event.tsp.TSPParameterDescription`) will consist of:

- A parameter ID.
- A label to be shown for the entry field.
- A description (to be provided as a help text).
- A type.
- An indicator if the parameter is optional or mandatory.
- A placeholder text to be used in the entry field if needed.
- A Java regular expression to be used to validate the user input.

The calling application can use the TSP info event to query the plug-in on the information needed. The TSP name and the validation text will be displayed in the validation window, together with a list

of entry fields defined by the validation credentials descriptions. If a registration URL is present, the application will display it, as part of a message where new users can create credentials if they are not yet registered.

TSP validation event

The `de.softpro.cirrus.plugins.event.tsp.TSPValidationEvent` describes the actual validation call. The input parameters are:

- **IN_CREDENTIALS**

The credentials, according to the information provided by the info event. Credentials are provided as a map with the parameter ID as a key and the value given for that parameter. To avoid exceptions, parameters should be validated with the regular expression returned by the info event.

- **IN_SETTINGS**

Optional. If present, the settings map will completely override the settings provided at plug-in instantiation via `IPlugin.injectPluginConfiguration`.

- **IN_SIGNATURE_TYPE**

The type of the digital signature requested. Currently BASIC, ADVANCED or QUALIFIED.

The output only provides a true / false condition depending on the validation outcome:

- **OUT_RESULT**

A boolean value indicating the result of the validation. True denotes a successful validation of the credentials for the specified signature type.

In case of processing errors, an appropriate exception will be thrown.

TSP post document signature event

The `de.softpro.cirrus.plugins.event.tsp.PostDocumentSignatureEvent` starts the signing process with the TSP provider. The input parameters are:

- **IN_CREDENTIALS**

The credentials, according to the information provided by the info event. Credentials are provided as a map with the parameter ID as a key and the value given for that parameter. To avoid exceptions, parameters should be validated with the regular expression returned by the info event.

- **IN_AUTHENTICATION_TOKEN**

An authentication token, if available.

In case the TSP uses a short lived authentication token to authenticate the service, the token can be passed here.

- **IN_SIGNATURE_TYPE**

The requested signature type (BASIC, ADVANCED or QUALIFIED).

- **IN_DOCUMENT**

The document to be signed (byte array).


- **IN_DOCUMENT_DESCRIPTION**

The description of the document.

- **IN_DOCUMENT_NAME**

The name of the document to be signed.

- **IN_REDIRECT_URL_OK**
URL to be redirected to if signing was successful.
- **IN_REDIRECT_URL_CANCEL**
URL to be redirected to if signing has been canceled.
- **IN_REDIRECT_URL_ERROR**
URL to be redirected to if a signing error occurred.
- **IN_SETTINGS**
Optional. If present, the settings map will completely override the settings provided at plug-in instantiation via `IPlugin.injectPluginConfiguration`.
- **IN_SIGNATURE_FIELD_NAME**
Optional. Should be used when signing a signature field using TSP. The name of the signature field that supports 'TSP' signing mode and that will be signed using the TSP plug-in.

 This parameter is added to be used as an identifier for the signature field in the PDF document and final signature can be applied to this field in the TSP plug-in.

The output data includes:

- **OUT_AUTHENTICATION_TOKEN**
A new authentication token after successful authentication, in case the TSP uses an authentication token mechanism.
- **OUT_DOCUMENT_VERIFICATION_URL**
The TSP document verification URL.
The Signing Client will redirect to this TSP URL letting the signer authenticate with the TSP service and sign the document.
- **OUT_SIGNATURE_PROCESS_TOKEN**
The signature token returned by the TSP after initiating the document signing. This token is used to identify this particular signing process.

TSP get document signature event

The `de.softpro.cirrus.plugins.event.tsp.GetDocumentSignatureEvent` is used to retrieve a signed document from the TSP previously sent for signing.

The input parameters are:

- **IN_AUTHENTICATION_TOKEN**
An authentication token, if available.
In case the TSP uses a short lived authentication token to authenticate the service, the token can be passed here.
- **IN_SETTINGS**
Optional. If present, the settings map will completely override the settings provided at plug-in instantiation via `IPlugin.injectPluginConfiguration`.
- **IN_SIGNATURE_PROCESS_TOKEN**
The signature token that identifies this signing process (received by `PostDocumentSignatureEvent`).

The output data includes:

- **OUT_AUTHENTICATION_TOKEN**
A new authentication token after successful authentication, in case the TSP uses an authentication token mechanism.
- **OUT_DOCUMENT**
The content of the signed document (byte array).
- **OUT_DOCUMENT_NAME**
The name of the document being returned.

TSP features with SignDoc

A TSP plug-in can also be used to sign a signature field. When a signer tries to sign a TSP signature field, the signer is directed to an external TSP service registered by the package creator for the signer. Based on the outcome of the signing process a message is displayed to the signer in the signing session console.

The parameter `IN_SIGNATURE_FIELD_NAME` is optional and should be used when signing a signature field using TSP. The name of the signature field that supports TSP signing mode and that will be signed using the TSP plug-in.

i This parameter is added to be used as an identifier for the signature field in the PDF document and final signature can be applied to this field in the TSP plug-in.

TSP-Loopback-Plugin is a test TSP plug-in which can be used for reference purposes.

To activate the TSP-Loopback-Plugin add

`de.softpro.cirrus.plugins.tsp.TSPLoopbackPlugin` to the existing `plugin.loadlist` configuration from the administrator's section. This plug-in can then be enabled to one account or all the accounts setting the configuration setting

```
plugin.enabled.TSP-Loopback-Plugin = true
```

A developer who wants to support the new features in their TSP plug-in should use the configurations below:

- **plugin.cfg.<pluginId>.tsp.signature.enabled**
Defines if the TSP plug-in supports TSP signing mode.
Default value: true

i It is necessary that the plug-in if defines this configuration should have a default value 'true' and have plug-in setting type `BOOLEAN`. The `clientCertificateRequired` option for a signer is mutually exclusive when used along with this configuration, that is, a signer with `clientCertificateRequired` set to true will not be able to have a signature field with TSP signing mode and vice versa even if this setting is enabled. It is necessary that this configuration allows view access to at least `ROLE_USER`, `ROLE_ADMIN` and `ROLE_SUPER`.

The configurations below are optional:

- **plugin.cfg.<pluginId>.tsp.field.redirect.message.enabled**

Defines if a redirect message should be available to the signer after initiating the TSP signing at the signature field level. If enabled, a redirect message is shown, else the signer is taken directly to the TSP signing process.

i If this configuration is not provided a redirect message will be displayed by default to the signer. It should have plug-in setting type BOOLEAN. It is necessary that this configuration allows view access to at least ROLE_SIGNER, ROLE_ADMIN, and ROLE_SUPER.

- **plugin.cfg.<pluginId>.tsp.field.overlay.image**

Overlay image for TSP signature field. Supported image formats are JPG and PNG.

i This configuration should support `MediaType.APPLICATION_OCTET_STREAM` and have plug-in setting type BINARY. It is necessary that this configuration allows view access to at least ROLE_USER, ROLE_SIGNER, ROLE_ADMIN, and ROLE_SUPER.

- **plugin.cfg.<pluginId>.tsp.document.redirect.message.enabled**

Defines if a redirect message should be available to the signer after initiating the TSP signing at the document level. If enabled, a redirect message is shown else the signer is taken directly to the TSP signing process.

i If this configuration is not provided, a redirect message will be displayed by default to the signer. It should have plug-in setting type BOOLEAN. It is necessary that this configuration allows view access to at least ROLE_SIGNER, ROLE_ADMIN, and ROLE_SUPER.

TSP document signing can also be switched on/off for a particular TSP plug-in in an account using the below configuration. However, this configuration is optional and if not defined the TSP document signing will be allowed by default for the TSP plug-in.

- **plugin.cfg.<pluginId>.tsp.document.enabled**

Defines if the TSP plug-in supports document signing using TSP.

i It is necessary that if the plug-in is defined, this configuration should have a default value 'true' and have plug-in setting type BOOLEAN. It is necessary that this configuration, if defined, allows view access to at least ROLE_USER, ROLE_ADMIN, and ROLE_SUPER.

Chapter 5

Email

This chapter describes the SMTP and S/MIME configuration.

SMTP configuration

Description

All possible SMTP related settings can be configured per SignDoc account or as a global database configuration using the Manage Client.

Prerequisites

- A valid SMTP server

Configuration

- mail.smtp.host
- mail.smtp.port
- mail.smtp.user
- mail.smtp.password
- mail.smtp.ssl.enable
- mail.smtp.from
- mail.smtp.starttls.enable
- mail.smtp.starttls.required
- mail.smtp.ssl.checkserveridentity
- cirrus.startup.email
- mail.smtp.auth
- mail.smtp.connectiontimeout
- mail.smtp.localhost
- mail.smtp.timeout
- mail.smtp.writetimeout
- mail.debug

Usage

Each SMTP configuration setting is evaluated first from the database for an account, then for global and if neither is set is finally retrieved from the system settings. The origin or the source of these settings (if they are retrieved from the database or are system settings) are logged at the 'DEBUG' level in the log messages.

The `cirrus.startup.email` configuration needs either the system or global configurations to work otherwise the startup email cannot be sent.

S/MIME configuration

Description

Outgoing emails can now be signed according to the S/MIME standard using a valid certificate and password. All certificates aligning with Java crypto architecture and compatible with JavaMail with a file format PKCS#12 extension are accepted.

Prerequisites

- The certificate must be in PKCS#12 format
- A valid certificate passphrase

Configuration

- `mail.s-mime.certificate`

Usage

The emails are signed if a valid certificate and password are configured. The certificate and password can be provided at the account level (evaluated first) or the global level (evaluated if account-specific settings are not available) configurations. If the certificate is about to expire a message is logged if the number of days to expiry is less than the limit defined in `client.account.certificate.expiry.warn.threshold` once a day when the first email is sent for an account and once a day for the global settings.

The message about the expiry of the certificate is recorded exponentially starting from number of days then hours and finally minutes. The number of days are logged if the days left are one or greater, when the expiry date is less than a day then the number of hours are displayed and finally when it is less than one hour the minutes are displayed.

Chapter 6

BankID

BankID is a citizen identification solution that allows companies, banks and government agencies to authenticate and conclude agreements with individuals over the internet.

To be able to use BankID's identification and signature features, users must install the BankID app on a mobile device or PC. They also need to order a BankID from their bank. The web service API of BankID can only be accessed with a valid SSL client certificate.

As part of BankID solution provided by SignDoc, users can authenticate and sign during the remote session of the signing client using the BankID application.

The following chapters describe the supported features and how users can configure and integrate the BankID service with SignDoc Standard Manage Client.

BankID Windows service configuration

The user must have a valid BankID SSL certificate along with the CA trust certificate. The path to these certificates must be mentioned in the configuration parameters. These configuration parameters are updated in the SignDocBankId.xml file that is part of the BankID Windows service provided to the customer.

All the following configuration parameters are a must for the BankID service integration to work.

- **service.context.url**
Externally accessible Context URL of the service.
- **server.port**
The server port to use for the service.
- **cirrus.url**
The connect URL of the SignDoc Standard cirrus context.
- **cirrus.extauth.shared.secret**
The shared secret.
- **bankid.url**
Web Service BankID URL or API.
- **bankid.cert**
Absolute path to the SSL certificate.
- **bankid.cert.password**
Passphrase for the SSL certificate.
- **bankid.ca**

Absolute path to the SSL Root CA.

After successful authentication or signing the client has an option to store the success response from the BankID REST API using the following configuration parameters. These configuration parameters are however optional.

- **bankid.response.storage.url**
Storage location for the BankID success response, should be a POST endpoint.
- **bankid.response.storage.url.secret**
'X-AUTH-TOKEN' header to be added for the storage 'POST' request to the 'bankid.response.storage.url'.

The maintenance of the storage URL is client's responsibility and is out of scope for SignDoc.

SSL configuration

The BankID Service can also be run with an SSL configuration by setting some configuration values. To activate the setting, they must be specified in SignDocBankId.xml.

In this file is an already commented HTTPS / TLS configuration section that can be uncommented and adjusted.

Example settings for using a PKCS#12 cert store:

```
<argument>--server.ssl.key-store-type=PKCS12</argument>  
<argument>--server.ssl.key-store=file:path_to_my_keystore.pfx</argument>  
<argument>--server.ssl.key-store-password=2beChanged!</argument>  
<argument>--server.ssl.key-alias=1</argument>
```

Installing the external SignDocBankIdService

Follow these steps to install the external SignDocBankIdService.

1. Unpack the SignDocBankIdService-x.x-windows.zip to any folder.
2. Configure the service in file SignDocBankId.xml (tailored as per above details).
3. Run the service_up.cmd.
4. Test if the service is started by calling localhost:6614 in the browser. You should see now SignDoc BankID in the browser.
5. To stop and remove the service run stopandremove.cmd.

Configuration parameters

BankID authentication

For the authentication to work the following configuration parameters must be set in the SignDoc Standard Manage Client:

- **cirrus.security.external.authentication.name**
For example BankID.
- **cirrus.security.external.authentication.sharedsecret**
This should be equal to the cirrus.extauth.shared.secret value in the SignDocBankId.xml file of the SignDocBankIdService.
- **cirrus.security.external.authentication.url**
This URL must end with suffix /extauth, such as <http://localhost:6614/extauth>.

The package creator should then assign this external authentication to the signer in the signer's settings while creating and adding the signer to the package.

The signer with external authentication enabled is directed to the external authentication when it starts its remote authentication session from the SignDoc Signing Client.

The following options are supported as part of BankID authentication:

- **Authenticate using Desktop**
To complete authentication on the desktop, the user needs to have a BankID application installed on the desktop where the authentication is performed. The user does not have to start the application manually for this to work. When the user clicks on this link the launch of the BankID app is triggered on the desktop, the user can then verify itself using its password. The user can verify or cancel the authentication session.
- **Authenticate using personal number**
Users can authenticate using their personal number. For this to work user should have the BankID app installed either on their phone or on their desktop and registered with their valid personal number. The app should be manually started on either of the devices. If the personal number is correct and the app is started the user is then taken to the verification page, where the user can verify their identity with the correct password. The user can verify or cancel the authentication session.
- **Authenticate using mobile device**
The user must install the BankID app with its BankID number registered. SignDoc supports QR code which is displayed on the authentication screen. The users must manually start their BankID app on their mobile device and scan the QR code. The user is then asked to enter their password and verify their identity. The user can verify or cancel the authentication session.

Upon successful verification the user is directed to the signing client with a success message. If the user's authentication fails due to wrong password or timeout issues, the user must start the authentication process again. If the user does not complete its authentication before the package expiration date, the session expires, and the user is directed to the 'failed' page. The user can cancel the authentication by **Cancel** on the authentication main screen or by clicking cancel on the BankID application. If the BankID application is not installed or if the personal number provided is incorrect or if the user does not proceed with the authentication and decides not to click **Cancel**, the authentication transaction times out after some time, the user is directed to the failed screen in all these cases.

BankID signing

Installing TSPBankID plugin in SignDoc

To use BankID signing feature the user should register BankID TSP plugin in SignDoc Standard managed client. To do so follow the steps below:

1. Sign in to the SignDoc Administration Center.
2. Go to **System Settings > Plugins**.
3. Under General add `de.softpro.cirrus.plugins.tsp.TSPBankID` to the configuration `plugin.loadlist`.
4. Under Enabled set the toggle switch for `plugin.enabled.TSPBankID` to 'on'.
5. Under Configuration, you can see TSPBankID plugin. When you click on the same you can modify the TSPBankID plugin related configurations.

The configuration under step 3 can only be handled from the Administration Center, the other steps can be managed by other administrators as well.

TSPBankID plugin configurations

The user can personalize the following configuration parameters for the TSPBankID plugin either from the system administrator or the account administrator.

- **plugin.cfg.TSPBankID.tsp_service.context.url**
Context URL of the BankID TSP Integration Service.
- **plugin.cfg.TSPBankID.tsp_service.display_nos_document_pages**
Number of document pages that will be displayed to the signer before proceeding to the BankID application for signing.
- **plugin.cfg.TSPBankID.tsp_service.help_text**
Additional TSP signing help text.
- **plugin.cfg.TSPBankID.tsp_service.provider_name**
The Provider name used in the user interface.
- **plugin.cfg.TSPBankID.tsp_service.shared_secret**
Shared secret of the Simple TSP Service.
- **plugin.cfg.TSPBankID.tsp_service.signing_text**
The text that is displayed to the signer before being redirected to the BankID service concerning the signing procedure.
- **plugin.cfg.TSPBankID.tsp_service.visible_data_template**
The text template that is displayed to the signer in the BankID application when signing. The following place holders can be used to set context sensitive information in the text: `{signer_name}`, `{document_name}`, `{document_page_count}`, `{signing_package_name}`, `{signing_package_owner_name}`, `{signing_package_owner_email}`.

The default texts configured for the above configurations are as below.

- `tsp_service.provider_name=Swedish BankID signing service`

- `tsp_service.signing_text`=Document signing for Swedish citizen's using the BankID is provided by Swedish BankID signing service. To use this online service, you need a BankID identity issued by your bank.
- `tsp_service.help_text`=BankID Document Signing uses the Swedish BankID federal service for signing documents.
- `tsp_service.visible_data_template`=I have received and read the document {document_name} with {document_page_count} page(s) as a recipient of the Signing Package {signing_package_name} sent by {signing_package_owner_name} ({signing_package_owner_email}). As signer {signer_name}, I will sign this document ({document_name}@@description@@@ [provided with following description: {document_description}]@@@description@@@) using BankID signing service.

** @@@description@@@ and the one enclosed in {} are just placeholders and are replaced with the exact text before displaying to the user.

To use the TSP signing the package creator must create a TSP package with "tspPluginId": "TSPBankID".

The signer with package created using the TSPBankID plugin is directed to the external TSP signing after it signs the document in the remote signing session using the SignDoc signing client. Based on the outcome of the TSP signing session the package status is updated.

The following options are provided as part of BankID signing.

- **Sign using Desktop**

To complete Signing on the desktop, the user needs to have a BankID application installed on the desktop it is performing the sign operation. The user does not have to start the application manually for this to work. When the user clicks on this link the launch of the BankID app is triggered on the desktop, the user can then sign using its password. The user is displayed verification data as configured in the configuration `plugin.cfg.TSPBankID.tsp_service.visible_data_template` in the BankID app. The user can sign or cancel the TSP sign session.

- **Sign using personal Number**

Users can sign using their personal number. For this to work user should have the BankID app installed either on their phone or on their desktop and registered with its valid personal number. The app should be manually started on either of the devices. If the personal number is correct and the app is started the user is then taken to the sign page, where the user can sign with correct password. User is displayed verification data as configured in the configuration `plugin.cfg.TSPBankID.tsp_service.visible_data_template` in the BankID app. User can sign or cancel the TSP sign session.

- **Sign using mobile device**

The user must install the BankID app with its BankID number registered. The SignDoc supports QR code and is displayed on the sign screen. The users must manually start their BankID app on their mobile device and scan the QR code. The user is displayed verification data as configured in the configuration `plugin.cfg.TSPBankID.tsp_service.visible_data_template` in the BankID app. The user is then asked to enter its password and sign. The user can sign or cancel the TSP sign session.

Upon successful signing the user is directed to the signing client with a success message. If the signing fails due to wrong password or timeout issues, the user must start the TSP signing process again. If the user does not complete its signing before the package expiration date, the

session expires, and the user is directed to the Signing Client with an error message. The user can cancel the signing by clicking **Cancel** on the 'Sign' main screen or by clicking **Cancel** on the BankID application. If the BankID application is not installed or if the personal number provided is incorrect or if the user does not proceed with the signing and decides not to click **Cancel**, the signing transaction times out after some time, the user is directed to the Signing Client with an error message in all these cases.

Audit

The start and end of the external authentication and TSP signing session is recorded in the audit trails. For a TSP signing a successful 'sign' records the BankID orderRef as well as the hash of the document for future reference.

Localization

The external authentication and the TSP signing are localized to support other locales and languages, English is supported by default.

References

For more details on BankID, see the information and guidelines on the BankID website for developer documentation.

Chapter 7

Tenant-specific URL

SignDoc supports a simplified login for systems with multiple accounts in different ways.

- **Option 1:** Preselect a SignDoc account with the `accountid` query parameter
- **Option 2:** Associate a DNS label with a SignDoc account

Preselect SignDoc account

This is option 1 for simplified login, see [Tenant-specific URL](#).

If a SignDoc installation has multiple accounts, it is possible to preselect the SignDoc account the user should use with the `accountid` query parameter. The `accountid` parameter takes the value entered in the **Account ID** field on the login page.

Example

```
https://myserver/cirrus?accountid=legal
```

Let's suppose the SignDoc installation has an account named `legal`, which matches with the `accountid` parameter. In that case, the user does not have to enter the account information on the login screen.

i This option requires no configuration, but the application context (in this case `/cirrus`) must be present.

Associate DNS label

This is option 2 for simplified login. See [Tenant-specific URL](#).

A SignDoc server administrator can associate a custom domain name (or domain label) with a SignDoc account. If a user opens an associated custom domain name in the browser, SignDoc will display the login page without the **Account ID** field. That is, the user only has to enter his credentials.


Example

```
https://legal.signdoc.mydomain.com
```

Prerequisites

- The custom domain name registered in the DNS must point to the SignDoc installation.

- The feature will only work if the GET request (that is, `https://legal.signdoc.mydomain.com`) contains the `X-Forwarded-Host` header. The `X-Forwarded-Host` is the de-facto standard for systems hosted behind a load balancer or reverse proxy.

 `cirrus.tenant.url.supported` is no longer used since SignDoc 3.2 release, the tenant URL feature is supported by default if the DNS label used is linked to a specific account. To not use this feature, the DNS label for the specific account should not be set.

How to configure

Let's suppose a customer wants to use the domain `legal.signdoc.mydomain.com` to preselect the account `legal` of the SignDoc installation on the login page.

A SignDoc server administrator navigates to the **DNS label** setting of the `legal` account in the SignDoc Administration Center. The **DNS label** setting is part of the **Account details** dialog box. The server administrator can set the **DNS label** either to the full custom domain name (that is, `legal.signdoc.mydomain.com`) or only to the DNS label of the domain (that is, `legal`). Using the full domain name is recommended since it eliminates potential name collisions.

References

For detailed information on the `X-Forwarded-Host` header, see the Mozilla website for developer documentation.

Chapter 8

Generative AI

SignDoc offers AI Tasks that use generative AI services.

AI functionality

To enable the AI Tasks, such as creating document descriptions, one must configure a supported AI provider.

SignDoc supports the following AI Providers:

- Azure OpenAI (<https://oai.azure.com>)
- OpenAI (<https://platform.openai.com>)

SignDoc offers these AI Tasks:

- Document Description

AI provider configuration

To use AI functionality, it's necessary to configure an AI provider in the Account Configuration. The configuration can be found here:

Settings > AI > AI providers

Azure OpenAI settings

Azure OpenAI is the OpenAI service offered in the Azure cloud.

- **signdoc.ai.providers.azure_openai.enabled:** Enables/Disables this provider.
- **signdoc.ai.providers.azure_openai.api_key:** The Azure OpenAI API key.
- **signdoc.ai.providers.azure_openai.endpoint:** The Azure OpenAI endpoint.
- **signdoc.ai.providers.azure_openai.deployment:** Name of the Azure OpenAI deployment.
- **signdoc.ai.providers.azure_openai.order:** If multiple AI Providers are enabled, the order attribute defines which provider has precedence. Precedence is defined by value. 0 has the highest priority. Larger values have lower priority.

OpenAI settings

The OpenAI settings are for the service offering of OpenAI.

- **signdoc.ai.providers.openai.enabled:** Enables/Disables this provider.

- **signdoc.ai.providers.openai.token:** The OpenAI API Token.
- **signdoc.ai.providers.openai.model:** The OpenAI Model to use.
- **signdoc.ai.providers.openai.order:** If multiple AI Providers are enabled the order attribute defines which provider has precedence. Precedence is defined by value. 0 has the highest priority. Larger values have lower priority.

AI Tasks

Document Description

The Document Description Task summarizes the document's content by analyzing its text. The words setting influences the length of the generated description, and language parameters determine the language of the description.

This Task can be executed automatically when uploading a new document or later in the user interface by manually clicking a button.

These Task settings can be set by an Account Administrator

- **signdoc.ai.tasks.generate.document.description.enable:** Enables/Disables the Task.
- **signdoc.ai.tasks.generate.document.description.auto:** Enables automatic AI document description on upload.
- **signdoc.ai.tasks.generate.document.description.words:** How many words should the generated description approximately contain?
- **signdoc.ai.tasks.generate.document.description.language.source:** In what language should the description be written:
 - USER: The language of the SignDoc user
 - DOCUMENT: The main language of the document
 - FIXED: A fixed language defined by setting:
signdoc.ai.generate.document.description.language.fixed
- **signdoc.ai.generate.document.description.language.fixed:** This language (use the English name of the language) will be used to generate the document description if signdoc.ai.generate.document.description.language.source is set to FIXED. Valid languages: Any language the AI model is capable of. Some examples are English, German, French, and Japanese.

Related API Endpoints are:

- Document Description:
GET /rest/v8/packages/PACKAGE_ID/documents/DOCUMENT_ID/ai/gen/desc
- Document Language:
GET /rest/v8/packages/PACKAGE_ID/documents/DOCUMENT_ID/ai/lang/main
- Enabled Status of AI task:
GET /rest/v8/ai/status/task/AI_TASK

Appendix A

External authentication using Microsoft and Google

SignDoc 3.4.0 introduces support for Microsoft and Google as OAuth authentication providers for external authentication of recipients. Package creators can include one or more authentication providers for recipients when assembling the signing package.

To enable Microsoft authentication in SignDoc, it must be installed with the following configuration settings:

```
oauth2.microsoft.client.registration.client-id: <Microsoft App Registration App ID>
oauth2.microsoft.client.registration.client-secret: <Microsoft App Registration Client Secret>
oauth2.microsoft.client.provider.authorization-uri: <Microsoft Authorization URI>
oauth2.microsoft.client.provider.token-uri: <Microsoft Token URI>
oauth2.microsoft.client.provider.user-info-uri: <Microsoft User Info URI>
oauth2.microsoft.client.provider.jwk-set-uri: <Microsoft JWK Set URI>
```

Replace <Microsoft App Registration App ID> and <Microsoft App Registration Client Secret> with the respective values obtained during the Microsoft application registration process. Also, replace <Microsoft Authorization URI>, <Microsoft Token URI>, <Microsoft User Info URI>, and <Microsoft JWK Set URI> with the appropriate URIs provided by Microsoft.

Similarly, to use Google authentication in SignDoc, it must be installed with the following configuration settings:

```
oauth2.google.client.registration.client-id: <Google Registration App Client ID>
oauth2.google.client.registration.client-secret: <Google Registration App Client Secret>
oauth2.google.client.registration.authorization-uri: <Google Authorization URI>
oauth2.google.client.registration.token-uri: <Google Token URI>
oauth2.google.client.registration.user-info-uri: <Google User Info URI>
```

Replace <Google Registration App Client ID> and <Google Registration App Client Secret> with the corresponding values obtained during the Google application registration process. Also, replace <Google Authorization URI>, <Google Token URI>, and <Google User Info URI> with the appropriate URIs provided by Google.

For detailed steps on obtaining these values, refer to the respective guides provided by Microsoft and Google.

Additionally, the following configuration settings facilitate the management of Microsoft and Google enablement at the account or system level within the Administrators section in SignDoc.

Use Google as an authentication provider


```
cirrus.security.external.authentication.oauth.google.enabled
```

If set to 'On', a package creator can assign Google as one of the authentication providers for a recipient, provided Google is registered in the SignDoc application.

Default value: On

Use Microsoft as an authentication provider

```
cirrus.security.external.authentication.oauth.microsoft.enabled
```

If set to 'On', a package creator can assign Microsoft as one of the authentication providers for a recipient, provided Microsoft is registered in the SignDoc application.

Default value: On

Validates recipient's email for OAuth authentication

```
cirrus.security.external.authentication.oauth.validate.recipient.email
```

If set to 'On', the recipient's email is validated against the email they used for OAuth authentication using Google or Microsoft. Authentication fails if the recipient attempts to authenticate using a different email. If set to 'Off', recipients can authenticate using any valid email address from the respective providers.

Default value: Off

Appendix B

Google Chrome Group Policy (GPO)

When using the Google Chrome Group Policy (GPO) some adjustments have to be made to support the SignDoc DeviceConnector browser extension. This extension is required to capture handwritten signatures from a signature pad.

Because the browser extension uses Chrome's Native Messaging API the GPO needs to be relaxed for two settings:

Allow user-level Native Messaging hosts

Enable user-level Native Messaging hosts via

```
Software\Policies\Google\Chrome\NativeMessagingUserLevelHosts = 1
```

Configure Native Messaging whitelist

Add deviceconnector to the whitelist via

```
Software\Policies\Google\Chrome\NativeMessagingWhitelist\1 = "  
de.softpro.sbpluginng "
```

Appendix C

Font substitution when converting from MS Word to PDF

SignDoc Standard uses a third-party vendor for handling PDFs and MS Word documents. The third-party library tries to find the closest match for a font when converting from MS Word to PDF. The fonts used to create the MS Word document should be either available or installed at the system level or in `signdoc_home`. They can also be provided in a separate custom directory and the path should be set in `client.manage.document.word-pdf.font-directory`. If a Word document is created using Calibri font, and if Calibri is not available (either in the system or in `signdoc_home`), the third-party library searches for all fonts in font sources with an exact font name and uses the closest font match available.

To help further debug and identify the fonts used, font info substitutions are printed at FINER levels.

Example 1

```
Font 'Segoe UI' has not been found. Using 'Noto Sans SC Black' font instead.  
Reason: first available font.
```

Example 2

```
Font 'Calibri' has not been found. Using 'Arial Unicode MS' font instead.  
Reason: font info substitution.
```

These info messages will help users to understand why the final PDF document sometimes appears different than the original MS Word document.