Kofax SignDoc Standard

Administrator's Guide

Version: 3.0.0

Date: 2021-08-16

KOFAX

Table of Contents

Preface

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

Related documentation

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

https://docshield.kofax.com/Portal/Products/SD/3.0.0-7s9x4v5c5f/SD.htm

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

- *Help for Kofax SignDoc Standard*
- *Help for Kofax SignDoc Standard Administration Center*
- *Help for Signing Documents with Kofax SignDoc*
- *Kofax SignDoc Standard Developer's Guide*
- *Kofax SignDoc Standard Installation Guide*
- *Kofax SignDoc Technical Specifications*

Training

Kofax offers both classroom and online training to help you make the most of your product. To learn more about training courses and schedules, visit the Kofax Education Portal on the Kofax website.

Getting help with Kofax products

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

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

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

The Kofax Knowledge Base provides:

- Powerful search capabilities to help you quickly locate the information you need.
  Type your search terms or phrase into the **Search** box, and then click the search icon.

- Product information, configuration details and documentation, including release news.

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

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

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

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

Definitions

- `INSTALLDIR` is the directory with the unpacked signdoc-standard-3.0.0.zip file. See Related documentation, *SignDoc Standard Installation Guide*, chapter "Quickstart procedure".
- `CIRRUS_HOME` and `SDWEB_HOME` are the directories of the web applications that compose SignDoc Standard.

  Note Starting with SignDoc Standard 2.1.0, these home directories are consolidated by default in one single directory. See Related documentation, *SignDoc Standard Installation Guide*, chapter "Directories".

Chapter 1

Licensing

License handling

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

For on-premise installations the SignDoc license file (KofaxSignDoc.key) will be provided by the Kofax 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

Starting with SignDoc 2.1.0.1 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:KofaxSignDoc
i:Manufacturer:Kofax Deutschland GmbH
i:Customer:Dummy corporation
i:Version:99
i:OS:all
a:product:unlimited
a:signware:unlimited
a:sign:2018-12-31
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 application-related license parameters:

`a:sign:2018-12-31` 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:F4D22WO65F` 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 system-wide license which is valid for all accounts. The included permissions 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 installed. No additional account-specific licenses are needed.

Important If a global license is installed once 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 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 count on 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 this case 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 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 Kofax if the installed license is invalid or corrupted for any reason. Since this is an unrecoverable problem, it requires a 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 an 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 only 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 production license.

Chapter 2

Logging

On-premise 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.

SignDoc Standard uses an inbuilt logging framework, which can be turned on/off using the configuration id 'signdoc.logger.handler.enabled'. It supports log levels (ALL, FINEST, FINER, FINE, CONFG, INFO, WARNING, SEVERE, OFF) which can be easily configured by the administrator user using the configuration id 'signdoc.logger.level'. The logging configurations can only be viewed and updated by an administrator user.

By default SignDoc Standard creates log files on disk. Cirrus-specific log files are written to SIGNDOC_HOME/logs/signdoc/signdoc.log by default, if the parent directory does not exist it will be created.

The directory containing the signdoc logs should solely be used to store the signdoc logs controlled by signdoc.logger.* configurations and should not contain any subdirectories, i.e. in this case, the "signdoc" folder inside the "logs" folder should not contain any more subdirectories and should not be referenced to write any other logs e.g. logs for the Tomcat server.

Cirrus writes its log by default into a primary file signdoc.log. A series of log files are created when the existing log file size exceeds the defined size limit. The log file with name signdoc.log always contains the most recent logs. The older log files exceeding the size limit are archived and have a timestamp appended to the filename for transparency. When a new log is written signdoc.log is first checked for the size limit defined in the configuration 'signdoc.logger.handler.logfile.maxsize', if the current log exceeds this size a new log file is created with name signdoc.log and the older file is archived at the same location and renamed as e.g signdoc_{TIMESTAMP}.log. The lowest possible allowed size is 10K kilobytes.

The user also has an option to control the maximum number of the log files to be archived at one time by using the configuration 'signdoc.logger.handler.logfile.maxnumber', when a new log line is written it checks if the total number of files including the new one does not exceed the defined limit. If the total number of log files exceeds this value, then all the oldest log files are deleted, only storing the total number of newest log files within the defined maxnumber limit.

This kind of file logging is intended only for on-premise usage.

Chapter 3

Configuration

Important SignDoc Standard before version 2.1.0 was mainly configured with the configuration file cirrus.properties. This file moved to `INSTALLDIR\_conf_templates\cirrus.properties` with version 2.1.0.

Since SignDoc Standard 2.1.0, it is highly recommended to use the file `INSTALLDIR\service_configuration.properties` (instead of cirrus.properties) whenever it is required to configure SignDoc with a configuration file. Configurations set in this file are applied as Java system property and have therefore highest precedence.

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

One configuration value can be set on following 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 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

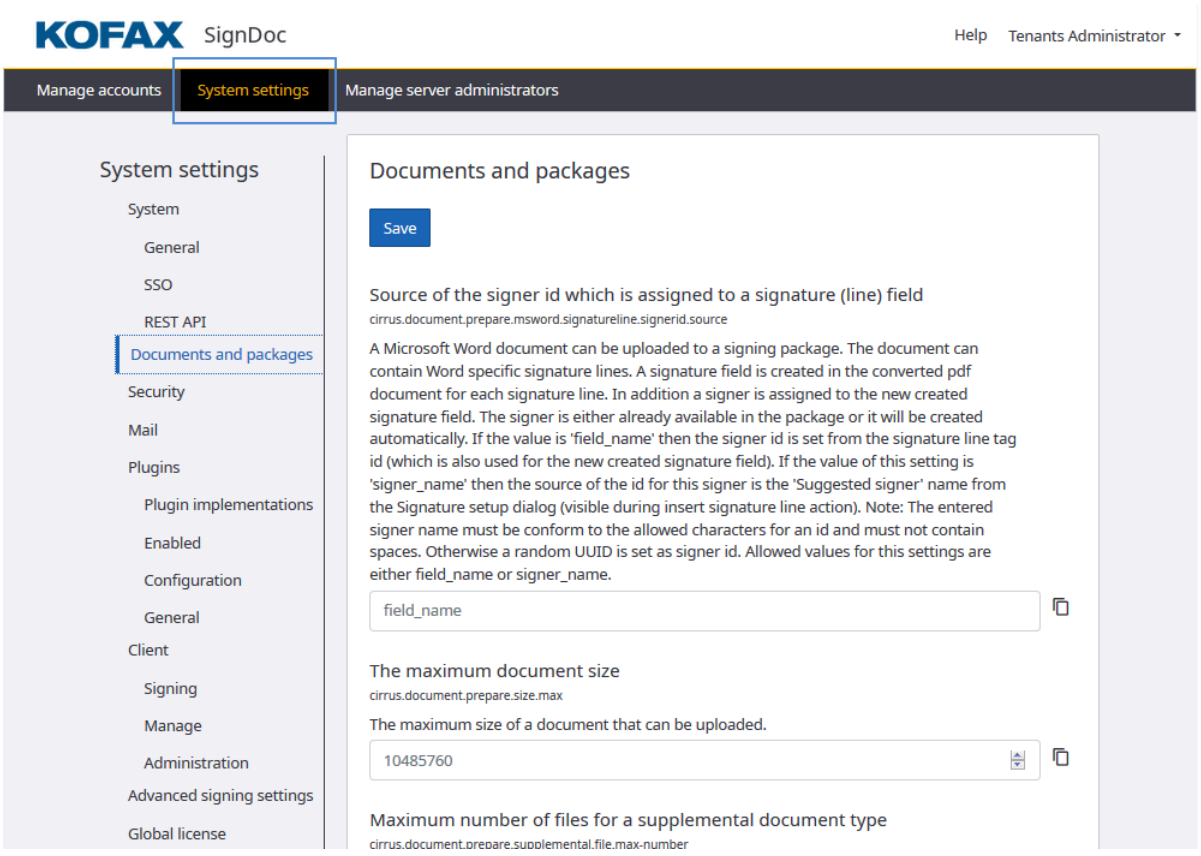
All configuration service values are set using the REST API. Both the Administration Center and the Manage Client provide user interfaces to view and edit configuration values. They use the underlying REST API to apply the changes.

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, plugins, security and signing.



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

Each configuration option will have:

- A title

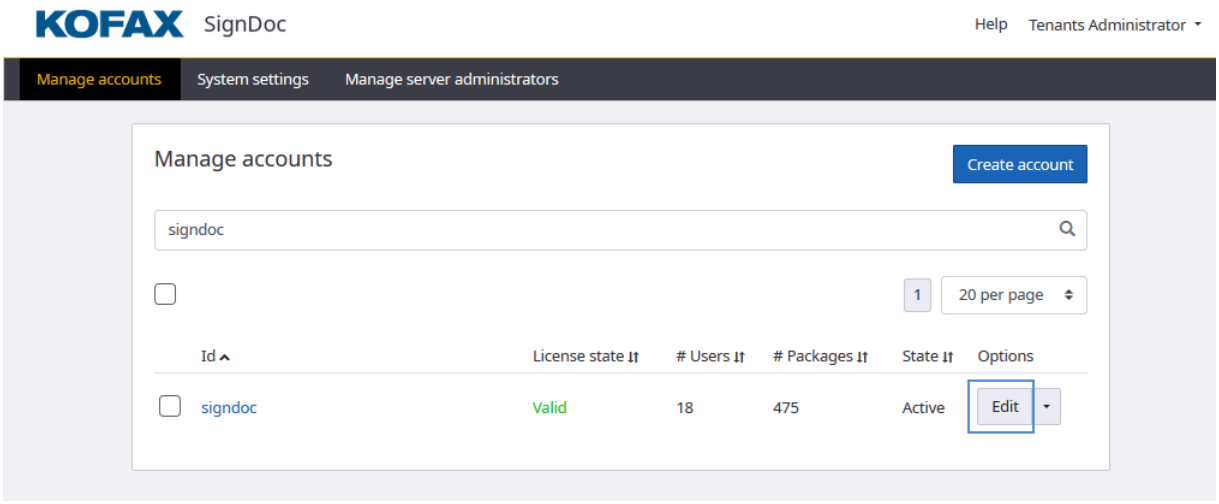
- The configuration option id
- A description of the configuration option

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.

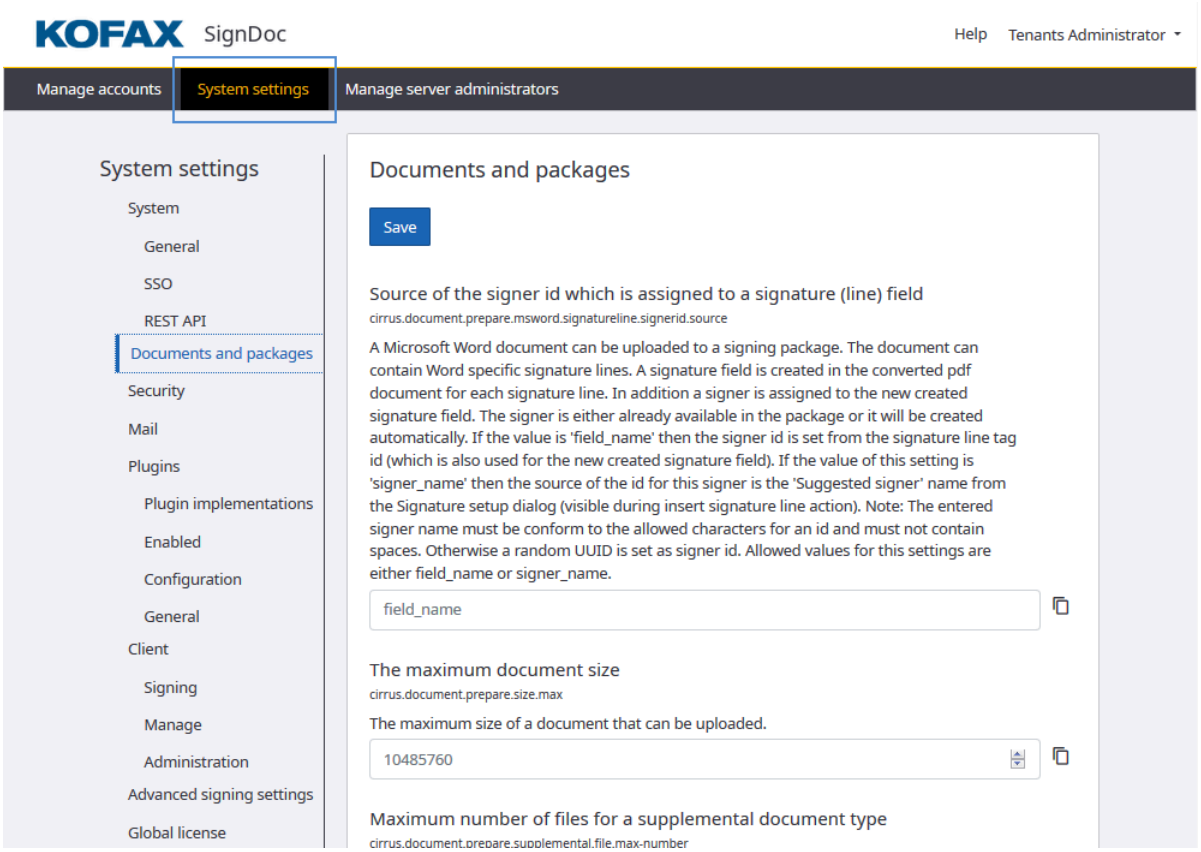
A configuration value will be validated by the client before it is used.

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

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

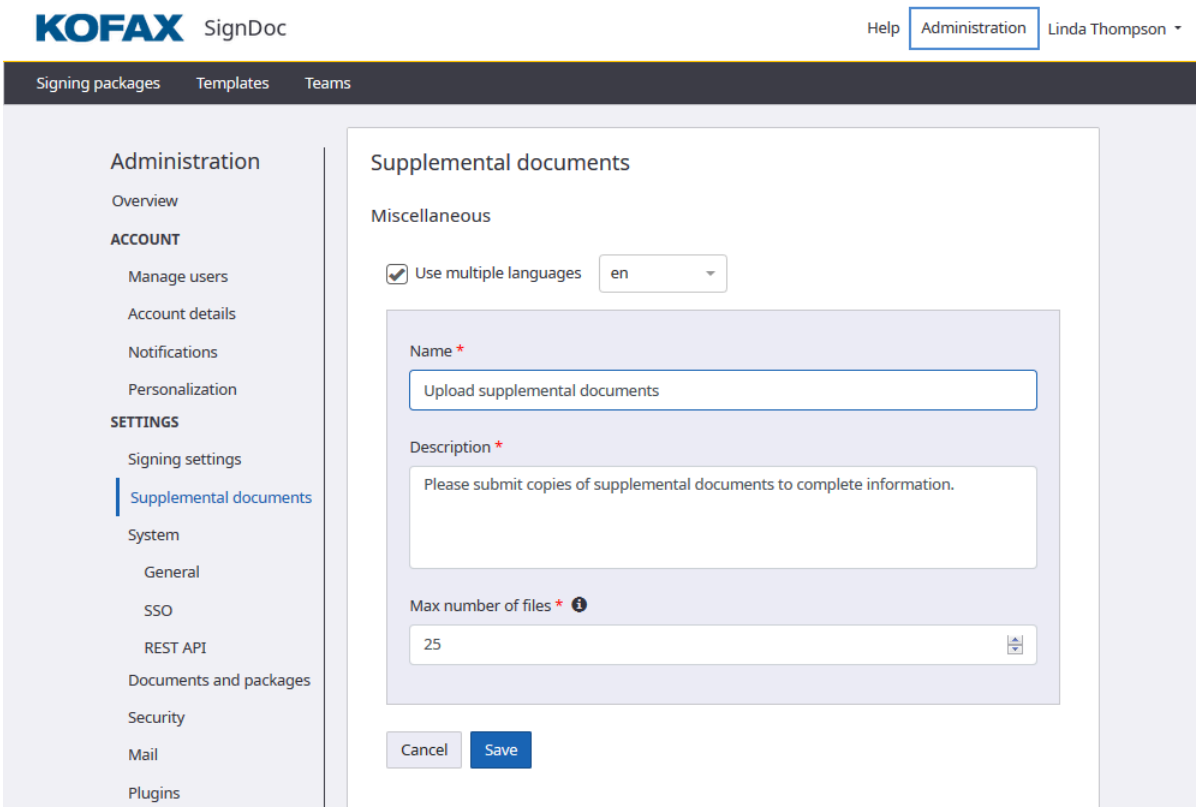


Clicking **Edit** will take you to a similar view as above, where the account-specific settings can be edited.



Configuration using the Manage Client

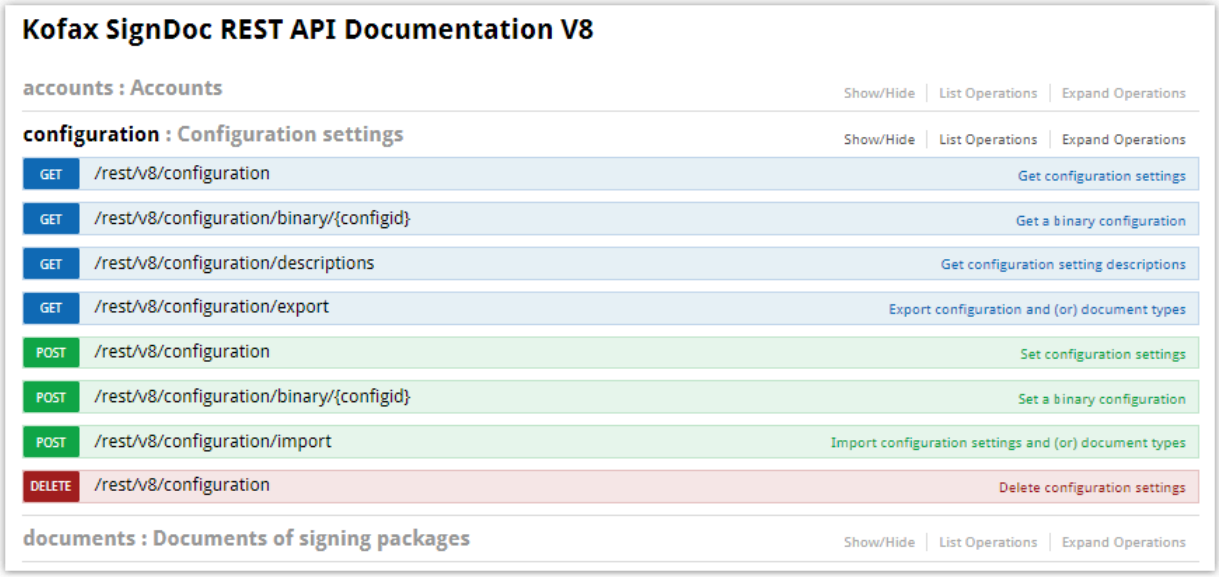
Account administrators will use the Manage Client to change configuration settings. To get to the configuration settings, click **Administration** in the title bar:



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.



Using the REST API is documented in the *SignDoc Standard Developer's Guide*, see Related documentation.

Configuration files

Important SignDoc Standard before version 2.1.0 was mainly configured with the configuration file cirrus.properties. This file moved to `INSTALLDIR\_conf_templates\cirrus.properties` with version 2.1.0.

Since SignDoc Standard 2.1.0, it is highly recommended to use the file `INSTALLDIR\service_configuration.properties` (instead of cirrus.properties) whenever it is required to configure SignDoc with a configuration file. Configurations set in this file are applied as Java system property and have therefore highest precedence.

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`
- **cirrus.tenant.url.supported**
  In the cirrus.properties configuration file this setting determines whether tenant-specific URL handling is supported or not. Supported values: true, false. Default: false

For information on LDAP properties, see *SignDoc Standard Installation Guide*, chapter "Authentication LDAP". See Related documentation.

The number of rest requests for a user in a time frame can be restricted using the below configuration options. Both the 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 a 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`

Configure the 'autoprepare' functionality

Important SignDoc Standard before version 2.1.0 was mainly configured with the configuration file cirrus.properties. This file moved to `INSTALLDIR\_conf_templates\cirrus.properties` with version 2.1.0.

Since SignDoc Standard 2.1.0, it is highly recommended to use the file `INSTALLDIR\service_configuration.properties` (instead of cirrus.properties) whenever it is required to configure SignDoc with a configuration file. Configurations set in this file are applied as Java system property and have therefore highest precedence.

During preparation of a signing package the user may choose to ‘autoprepare’ 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 top/left margin. If a field would exceed the right margin it will be placed in a next row. If there is no space left on the page, an exception is thrown.

Properties used for configuring the 'autoprepare' functionality

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

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

Important SignDoc Standard before version 2.1.0 was mainly configured with the configuration file cirrus.properties. This file moved to `INSTALLDIR\_conf_templates\cirrus.properties` with version 2.1.0.

Since SignDoc Standard 2.1.0, it is highly recommended to use the file `INSTALLDIR\service_configuration.properties` (instead of cirrus.properties) whenever it is required to configure SignDoc with a configuration file. Configurations set in this file are applied as Java system property and have therefore highest precedence.

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 (e.g 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 values

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.

- **List of configurable languages**
  cirrus.config.locales
  The comma separated list of locale strings for storing and retrieving the locale-specific configuration values, like email subject and body. A locale string must be a valid IETF BCP 47 language tag, e.g. en-US, see https://tools.ietf.org/html/bcp47.
- **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://tools.ietf.org/html/bcp47.
  Default value: en
  Example: en for English or pt-BR for Brazilian Portuguese.
- **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 e.g. key=value. 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. 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. User can provide a complete set or a subset for the localization of a specific language.
  Default value: sc_language.json

  Note
  - 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.

- **Prefer officially supported languages by Kofax**
  cirrus.general.localization.prefer.kofax.languages
  Kofax 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 Kofax 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

- **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.
- **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).
- **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).

Single Sign-on settings

- **Single Sign-on authentication module URL**

  cirrus.sso.auth.module.url

  The URL of the Single Sign-on authentication module. Examples: http://localhost:6612, https://sso.mysigndocserver.com
- **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.
- **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.
- **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.

- **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).
- **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'.
- **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 /v6/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
- **Timeout event request**

  cirrus.rest.event.request.timeout

  The maximum wait time in seconds (for long polling) if one or more events are requested via REST API.
- **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 auto-prepare option for documents**

  cirrus.rest.expresspackage.auto-prepare

  If set to true, 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.
- **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.

- **Automatic field masking**

  cirrus.rest.field.masking.automatic

  If set to 'on', fields that cannot be edited by the current signer or reviewer 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: 'off'
- **Automatic field masking for reviewers**

  cirrus.rest.field.masking.reviewer

  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: 'off'
- **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.
- **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.
- **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
- **REST API V6 default resolution**
  cirrus.rest.v6.default.resolution
  Defines the default resolution of the coordinates passed to the REST API V6 for the POST /package resource.

Logging settings

- **SignDoc log console logging**
  signdoc.logger.handler.console.enabled
  Enables console logging via the SignDoc log console logging. If 'on', SignDoc will print out qualified log lines on the console. Default value: On
- **SignDoc log Date format**
  signdoc.logger.handler.date.format
  Sets the format of the timestamp used in the SignDoc log file.
- **Enable SignDoc log**
  signdoc.logger.handler.enabled
  Controls the SignDoc log. If 'off', the following settings are completely ignored: signdoc.logger.handler.file.enabled, signdoc.logger.handler.logfile, signdoc.logger.handler.date.format, signdoc.logger.handler.console.enabled, signdoc.logger.handler.logfile.maxsize, signdoc.logger.handler.logfile.maxnumber. Default value: On
- **Enable SignDoc file logging**
  signdoc.logger.handler.file.enabled
  Controls the SignDoc file logging. If 'on', SignDoc will print out qualified log lines in the specified log file. See signdoc.logger.handler.logfile. Default value: On
- **SignDoc log file name**
  signdoc.logger.handler.logfile
  Defines the file used by the SignDoc log. The SignDoc process must be able to write to the file. If the file does not exist, the application will try to create the file as well as all needed parent directories.

  Note For this feature to work the best, the directory with the signdoc log files should not contain any other sub directories and should be used solely to store signdoc logs.

- **SignDoc log files maximum number**
  signdoc.logger.handler.logfile.maxnumber
  Defines the maximum number of SignDoc log files that can be stored or maintained. If the total number of SignDoc log files exceeds this set number all the previous log files are deleted. The lowest possible allowed size is 10K kilobytes.
- **SignDoc log file maximum size**
  signdoc.logger.handler.logfile.maxsize
  Defines the maximum size of a SignDoc log file in kilobytes, if the SignDoc log file size is more than the set limit a new log file is created for future logs.

- **SignDoc log logging level**
  signdoc.logger.level
  Sets the system wide logging level for all instances. Warning: A level lower than INFO (CONFIG, FINE, FINER, FINEST, ALL) can slow down the system and reduce performance. Valid log levels: OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL.
- **Enable external SignDoc log output**
  signdoc.logger.parent_handlers.enabled
  If 'on', all log lines will also sent to the standard java logging framework. Default value: On

Documents and packages

This section lists the settings related to documents and packages.

- **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.
- **The maximum document size**
  cirrus.document.prepare.size.max
  The maximum size of a document that can be uploaded.
- **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.
- **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.
- **Controls if documents are tested after upload**
  cirrus.document.prepare.validate.on.upload
  If 'On' documents lacking advanced compliance or having known vulnerabilities will be rejected. Default value: On

- **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.
- **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.
- **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
- **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 by uploading a flat PDF document. If the uploaded data is not usable it will be rejected.
- **Lock all interactive fields of finished signing package documents**
  cirrus.document.signing.final-package.lock-fields
  If a signing package is completed and this setting is enabled, all interactive fields of all documents will be locked to prevent further modifications. Default value: Off
- **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
- **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.
- **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.

- **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.
- **Enable delegate signing**
  cirrus.package.delegate.enabled
  Controls the delegation of a signing session for a recipient. If set '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 '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
- **Enable in-person signing**
  cirrus.package.inperson.enabled
  Controls the in-person signing. If '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 'on' the user can overwrite the in-person signing option. If 'off' user is not allowed to do any changes. Default value: On

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.
- **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).
- **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 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).

- **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.

- **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.

- **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).

- **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).

- **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

Mail

This section lists the mail related settings.

S/MIME related settings

- **S/MIME Certificate**

  mail.s-mime.certificate

  Certificate which will be used to sign the MIME data.

General mail 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://tools.ietf.org/html/bcp47. Example: en for English or pt-BR for Brazilian Portuguese.
- **Delegate link in email**

  cirrus.mail.delegate.enabled

  If '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 send 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.
- **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.
- **Final document notification**

  cirrus.mail.finaldocument.notification.enabled

  Email notifications for the final document. Default value: On

  Description

  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 behaviour:

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

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

  Important A configuration setting in the configuration database takes precedence over a setting in cirrus.properties file.
- **Send a mail if a user password has been changed**

  mail.enabled.password.changed

  Send a notification mail to the user if his password has been changed.
- **Access code text for missing delivery plugin**

  mail.message.accesscode.error

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

- **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.
- **Access code text for delivery by plugin**
  mail.message.accesscode.plugin
  The access code text for delivery by plugin. The only placeholder allowed is %%NOTIFICATIONPLUGINTYPE%% which will query the plugin type used.
- **Account disabled email body**
  mail.message.account.disabled.body
  The body for account disabled emails.
- **Account disabled email subject**
  mail.message.account.disabled.subject
  The subject line for account disabled emails.
- **User invitation email body**
  mail.message.account.invited.body
  The body for user invitation emails.
- **User invitation email subject**
  mail.message.account.invited.subject
  The subject line for user invitation emails.
- **Changed password email body**
  mail.message.changed.password.body
  The body for changed password emails.
- **Changed password email subject**
  mail.message.changed.password.subject
  The subject line for changed password emails.
- **Document copy email body**
  mail.message.email.me.a.copy.body
  The body for document copy emails.
- **Document copy email subject**
  mail.message.email.me.a.copy.subject
  The subject line for document copy emails.
- **Password forgotten email body**
  mail.message.forgotten.password.body
  The body for password forgotten emails.
- **Password forgotten email subject**
  mail.message.forgotten.password.subject
  The subject line for password forgotten emails.
- **Reviewer complete email body**
  mail.message.inform.owner.about.reviewer.complete.body
  The body for owner email after reviewer complete.

- **Reviewer complete email subject**
  mail.message.inform.owner.about.reviewer.complete.subject
  The subject line for owner email after reviewer complete.
- **Signer complete email body**
  mail.message.inform.owner.about.signer.complete.body
  The body for owner email after signer complete.
- **Signer complete email subject**
  mail.message.inform.owner.about.signer.complete.subject
  The subject line for owner email after signer complete.
- **Package complete owner email body**
  mail.message.package.complete.owner.body
  The body for owner email after package complete.
- **Package complete owner email subject**
  mail.message.package.complete.owner.subject
  The subject line for owner email after package complete.
- **Package complete recipient email body**
  mail.message.package.complete.recipient.body
  The body for recipient email after package complete.
- **Package complete recipient email subject**
  mail.message.package.complete.recipient.subject
  The subject line for recipients email after package complete.
- **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.
- **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.
- **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.
- **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.
- **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.

- **Signer declined email body**
  mail.message.rejected.body
  The body for signer declined emails.
- **Signer declined email subject**
  mail.message.rejected.subject
  The subject line for signer declined emails.
- **Reminder email notification body**
  mail.message.reminder.body
  The body for reminder email notifications.
- **Reminder email notification subject**
  mail.message.reminder.subject
  The subject for reminder email notifications.
- **Password reset email body**
  mail.message.reset.password.body
  The body for password reset emails.
- **Password reset email subject**
  mail.message.reset.password.subject
  The subject line for password reset emails.
- **Reviewer notification email body**
  mail.message.reviewing.body
  The body for reviewer notification emails.
- **Reviewer notification email subject**
  mail.message.reviewing.subject
  The subject for reviewer notification emails.
- **Custom reminder email body**
  mail.message.send.message
  The body for custom reminder emails.
- **Custom reminder with link email body**
  mail.message.send.message.with.link
  The body for custom reminder with link emails.
- **Signer notification email body**
  mail.message.signing.body
  The body for signer notification emails.
- **Signer notification email subject**
  mail.message.signing.subject
  The subject for signer notification emails.
- **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 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.
- **Team invitation email body**
  mail.message.user.add.to.team.invited.body
  The body for team invitation emails.
- **Team invitation email subject**
  mail.message.user.add.to.team.invited.subject
  The subject line for team invitation emails.
- **Mail debug**
  mail.debug
  Sets debug for mail, 'off' by default. Setting the value to 'on' will cause JavaMail to print debugging messages as it attempts to load each configuration file. Default value: Off

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
  Port of the SMTP server.
- **User**
  mail.smtp.user
  Username 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.
- **Start TLS enabled**
  mail.smtp.starttls.enable
  Enables or disables SMTP Start TLS setting. Default value: On
- **Start TLS required**
  mail.smtp.starttls.required
  Enables or disables SMTP Start TLS required setting. Default value: On

- **SSL Check Server identity**
  mail.smtp.ssl.checkserveridentity
  Enables or disables SMTP check server identity setting. Default value: On
- **SSL enabled**
  mail.smtp.ssl.enable
  Enables or disables use of SSL for SMTP connections. 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 username and password. If username 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.

Plugins

This section lists the general settings related to plugins.

Plugin implementations

This section contains the implemented plugins.

Enabled

This section contains the enabled plugins.

Configuration

In this section the enabled plugins can be configured.

General

- **Plugin directory**
  plugin.directory
  The directory where plugins will be located (in addition to the CLASSPATH).
- **Plugin load list**
  plugin.loadlist
  The list of plugin ids to be loaded. Ids must be separated by ','.

Client

This section lists the client related settings.

Signing Client related settings

- **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.
- **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 the e-sign consent text (max 2000 chars).
- **Requirement for GDPR consent**
  client.signing.gdpr.required
  The requirement for displaying the GDPR (EU 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 (EU General Data Protection Regulation) text which must be agreed by a recipient before signing or reviewing a signing package.
- **The external GDPR policy URL**
  client.signing.gdpr.url
  The custom GDPR (EU General Data Protection Regulation) URL which is provided as link in the Signing Client in addition the GDPR data protection 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
- **Enable Signing Client online help**
  client.signing.general.onlinehelp.visible
  If '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 'on', it is stored in the browser persistently. So the signer can re-use the image also in his following signing sessions. If '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
- **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
- **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 footer**
  client.signing.view.general.footer.visible
  Defines if the footer is shown in the signing client. Default value: On
- **Show header**
  client.signing.view.general.header.visible
  Defines if the header is 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 wizard-steps**
  client.signing.view.general.wizardsteps.visible
  Defines if the wizard-steps are shown in the signing client. 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 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 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 "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
- **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
- **The external finish URL**
  client.signing.view.finish.url
  The custom URL which is called when a signing session is finished by the remote signer. If no URL is provided the default finish page is displayed in the signing-client. Default value: On
- **The external resume later URL**
  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 could be configured by placeholders for accountId, packageId and signerId that are automatically replaced by data (ex: http://www.myserver.com/mypage.html?accountid=$ACCOUNT_ID&packageid=$PACKAGE_ID&signerid=$SIGNER_ID). If no URL is provided, the default finish page is displayed in the Signing Client.
- **The session expired URL**
  client.signing.view.session_expired.url
  The custom URL which is called when a signing session of a remote signer expired. It could be configured by placeholders for accountId, packageId and signerId that are automatically replaced by data (ex: http://www.myserver.com/mypage.html?accountid=$ACCOUNT_ID&packageid=$PACKAGE_ID&signerid=$SIGNER_ID). If no URL is provided, the default session expired page is displayed in the Signing Client.
- **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

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.
- **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. By default, fonts from the system and 'SIGNDOC_HOME' are used.

- **Skip the landing page**
  client.general.skip.landing
  Skips the initial landing page displayed before the login form. Default value: On
- **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
- **Enable Manage Client online help**
  client.manage.general.onlinehelp.visible
  If '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 after a signing package expires. The value 0 means that a signing package never expires. The maximum supported value is 365 days.
- **Recipients must be selected**
  client.manage.restrict.recipients.input
  If set '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
  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
- **Enable Administration Center online help**
  client.admin.general.onlinehelp.visible
  If '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.

- **2FAOn access code length**
  cirrus.security.2fa.accesscode.length
  The length of the generated random access code for the two factor authentication.
- **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.
- **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 '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).
- **Signature (certificate) type**
  client.signing.tsp.signature.type
  Signature (certificate) type which is needed for signing (BASIC, ADVANCED or QUALIFIED).

- **Device Connector URL**
  client.signing.deviceconnector.url
  The URL of the Device Connector server.
- **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

Expert configuration

The following configuration should only be used, if there are special requirements for the digital signing process:

- plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters
- plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters
- plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DSS
- plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters.DTS
- plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DTS

The format is XML and the DTD can be found here:

```
INSTALLDIR/signdoc_home/conf/SignDocParameters.dtd
```

More detailed documentation concerning the listed SignDoc SDK functions below can be found in the SignDoc SDK documentation. See Related documentation.

The evaluation and mapping of the setting values in SignDoc Standard goes like this:

```
[pseudo code]

if (plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters != null) {
   plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters =>
 de.softpro.doc.SignDocSignatureParameters
}
if (plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters != null) {
  plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters =>
 de.softpro.doc.SignDocVerificationParameters
}
// Add regular signature using the settings above
SignDocDocument.addSignature(SignDocSignatureParameters, SignDocVerificationParameters)

if (plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DSS != null) {
  // Extend the validity of the signatures in a PDF document (long term validity, LTV,
 PAdES-LTA)
  plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DSS =>
 de.softpro.doc.SignDocVerificationParameters
  SignDocDocument.updateDSS(v2, SignDocVerificationParameters);
}
if (plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters.DTS != null ||
 plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DTS != null) {
```

```
  // Add a document time stamp
  plugin.cfg.SignDocDefaultSigningHandler.sdsdk.SignatureParameters.DTS =>
 de.softpro.doc.SignDocSignatureParameters
  plugin.cfg.SignDocDefaultSigningHandler.sdsdk.VerificationParameters.DTS =>
 de.softpro.doc.SignDocVerificationParameters
  SignDocDocument.addSignature(SignDocSignatureParameters,
 SignDocVerificationParameters)
}
```

Chapter 4

Plugins

Plugin handling

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

There are 2 kinds of plugins: core plugins and custom plugins

- **Core plugins** are always loaded and are enabled by default for all accounts.
- **Custom plugins** 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 plugin

The required resources to implement a plugin can be found in the directory signdoc_home/interfaces/plugins/

This directory contains the required components

- Documentation
  - SignDoc Standard plugin definitions: cirrus-plugin-definitions-VERSION-javadoc.zip
  - Plugin 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 plugins.

How to deploy a plugin

The CIRRUS_HOME (signdoc_home) directory of SignDoc Standard contains a directory where all plugins can be installed:

signdoc_home/plugins

General directory structure

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

There is one plugin directory that is treated in a special way: default/. This directory is always present and has the highest priority amongst all plugin directories. If unsure, plugins should be deployed in this i.e. the 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/**
  The classes directory can hold single classes and resources and overrides the same classes of a lib directory in the same plugin 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/**
  The lib directory can hold jar files.

Important

- 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 plugin directory has the highest priority. I.e. a class deployed in the default/ directory will always be preferred over a class with the same class name in a custom plugin directory (e.g. a_custom_plugin/).
- The priority of the plugin directories other than the default/ plugin directory is undefined. I.e. if a class with the same classname is present in different directories it is undefined, which of these classes will be used.
- A plugin must be deployed with all required dependencies.
- If a new plugin is deployed, the files are immediately available for the SignDoc Standard application. A restart of the server application is not required.
- If plugin directories and/or plugin 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/ plugin 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.

Plugin administration

In the Administration Center

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

In the Manage Client

Plugins 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.

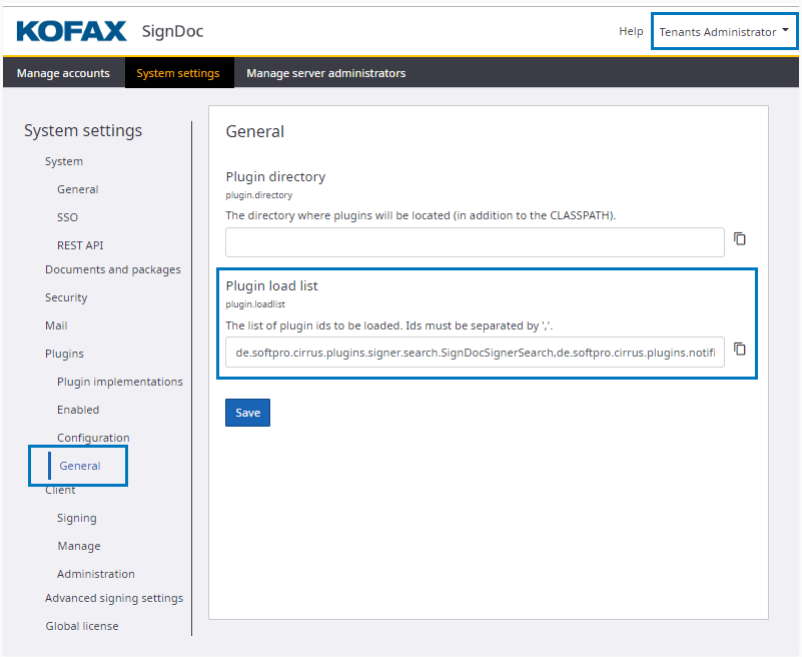
Tasks that can be done as account-specific administration are:

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

Load a plugin (only in the Administration Center)

On the **System Settings** menu, click **Plugins** > **General**.

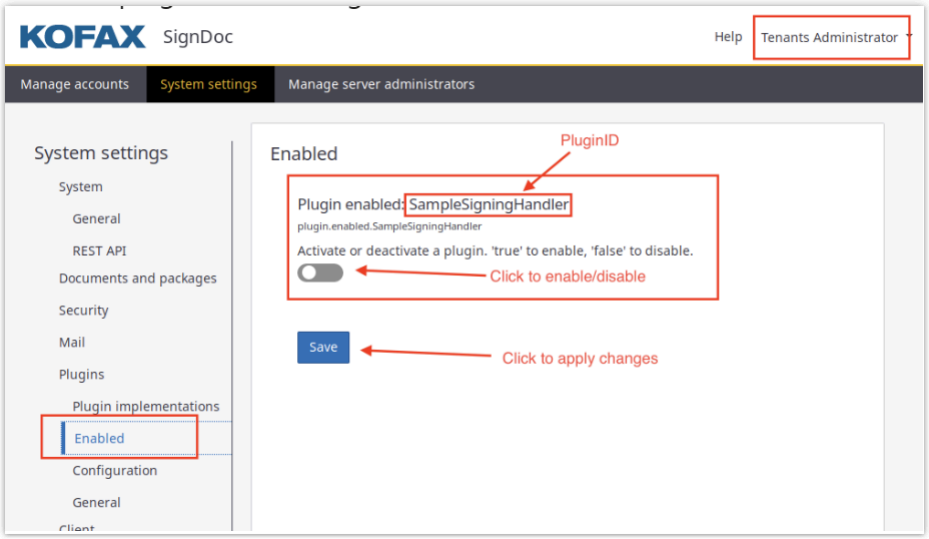
Add the class name of the plugin to the **Plugin load list** and save the settings.



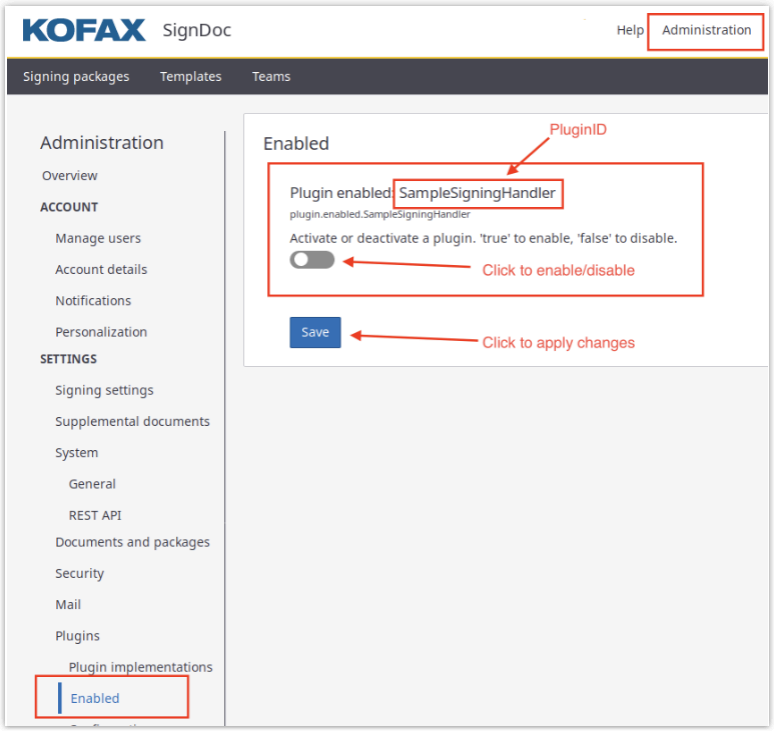
Enable or disable a plugin (Administration Center and account administration in Manage Client)

Note Before a plugin can be enabled, it must be loaded.

In the Administration Center, on the **System Settings** menu, click **Plugins** > **Enabled**. Enable or disable the plugin by clicking the control and then save the settings.



In the Manage Client, on the **Administration** menu, click **Plugins** > **Enabled**. Enable or disable the plugin by clicking the control and then save the settings.

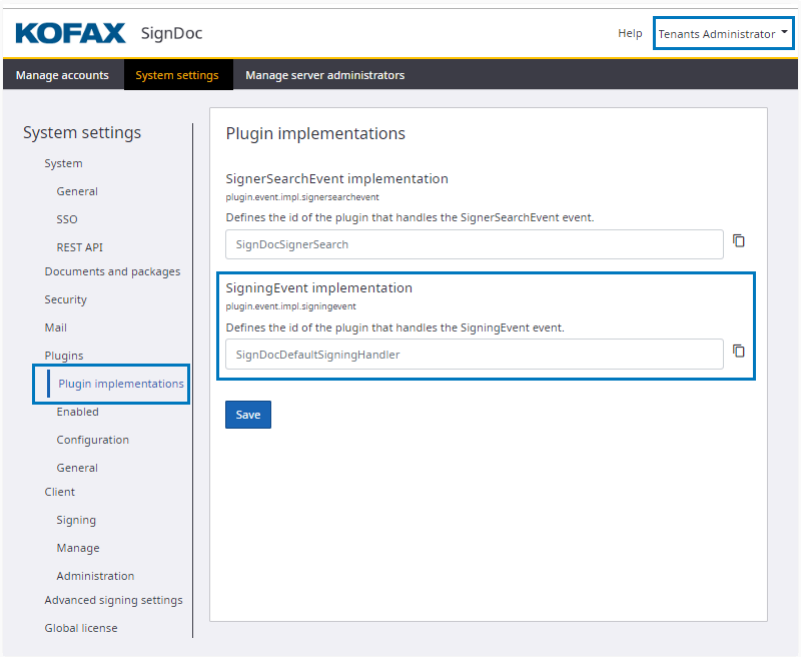


Assign a plugin implementation to a specific event

Note Before a plugin can be assigned to an event, it must be loaded.

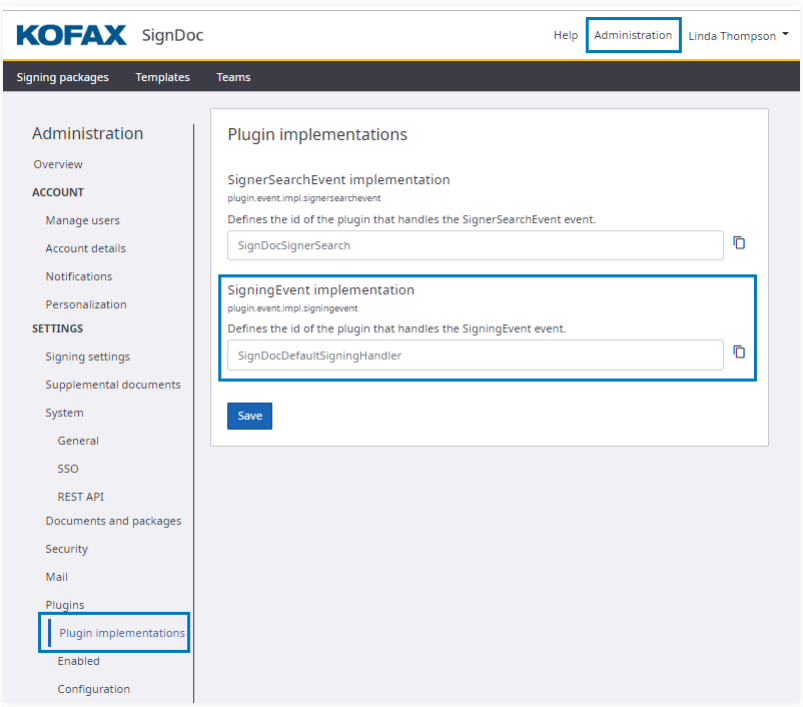
In the Administration Center, on the **System Settings** menu, click **Plugins** > **Plugin implementations**.

Enter the plugin id and then save the settings.



In the Manage Client, on the **Administration** menu, click **Plugins** > **Plugin implementations**.

Enter the plugin id and then save the settings.



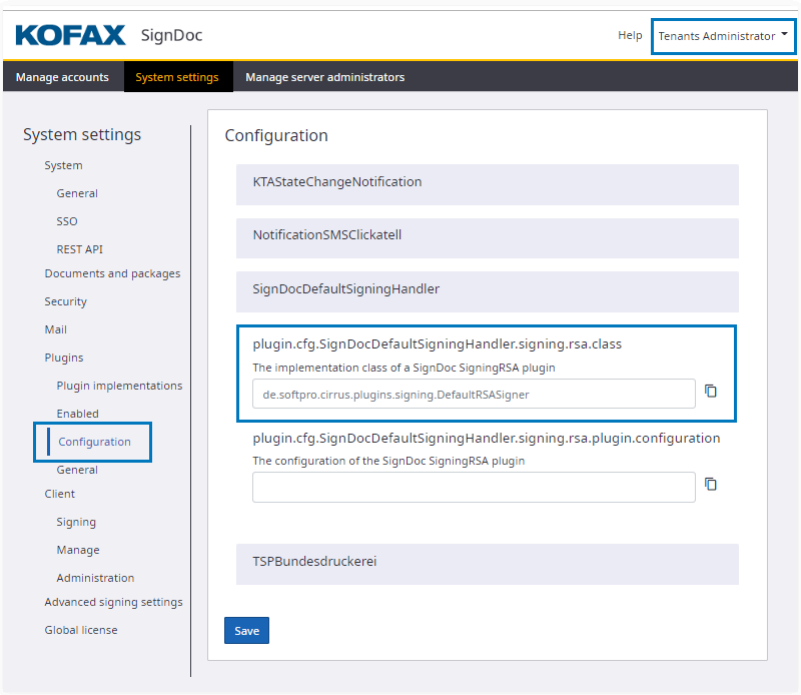
After a plugin has been enabled, it must be assigned to a specific event to modify or extend the SignDoc Standard behavior.

Configure plugins

In the Administration Center, on the **System Settings** menu, click **Plugins** > **Configuration**.

Select a plugin id from the list.

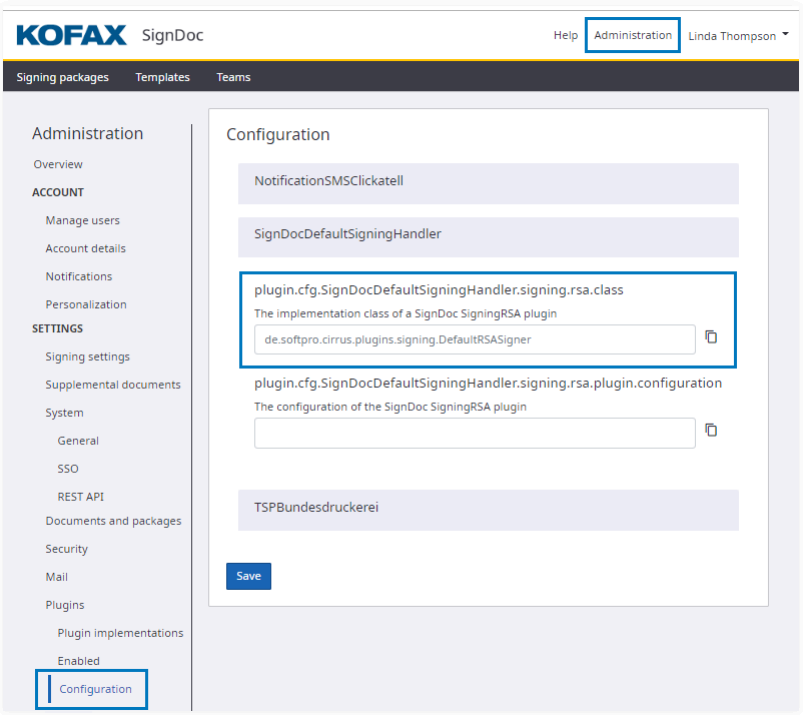
Set the value for the implementation class and save the settings.



In the Manage Client, on the **Administration** menu, click **Plugins** > **Configuration**.

Select a plugin id from the list.

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



Plugins can provide specific configuration options that can be configured in the Administration Center and in the account administration of the Manage Client.

Plugin development

Plugin interface

SignDoc Standard plugins support the ‘event’ plugin interface. In addition to that, they support the definition of plugin configuration data and the parameter description.

A plugin must implement the following interfaces.

IPlugin

The de.softpro.sppluginif.IPlugin interface defines the general plugin parameters. This includes the:

- Plugin id – used to identify the plugin. This has to be unique.
- General plugin information – such as vendor, description, copyright.
- Error message information – provide localized information for a particular error.
- Injection point for plugin configuration information – is called by the application with the plugin configuration data when the plugin is instantiated.

IEventPlugin

The de.softpro.sppluginif.IEventPlugin interface defines the calling procedure for event based plugins. A plugin 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 plugin can make its configurable parameters known to the application.

The application will use *getSettingDescriptions* to query what configuration settings the plugin supports. The plugin 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 plugin configuration. It is also used by the configuration service to validate entries.

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

Plugin implementation

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

For exceptions thrown by the plugin, 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 plugins

- For a plugin to be used, the server administrator (role SUPERUSER) has to add a plugin to the loadlist. After the loadlist is changed with the configuration service, the application will dynamically reload the plugins in the loadlist.
- For each plugin on the loadlist, the application will use the *IConfigurablePlugin* interface, if it is implemented, to query the configuration information that the plugin 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 plugin id.
- The server administrator and/or the account administrators can then enable the plugin globally or for specific accounts by setting the *plugin.enabled.<pluginId>* configuration setting to true. For every enablement an instance of the plugin will be created. If the plugin is enabled globally, only one instance will be created that is accessible for all accounts. If the plugin 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 plugin. 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 plugin on an account-specific basis.
- The plugin can now be used. The application will post supported events to the plugin using *eventCallback* from the *IEventPlugin* interface.

Signing plugin

SigningEvent plugin description

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

For a simple sample implementation, see Minimal SigningEvent implementation.

SignDoc Standard provides a default SigningEvent plugin. The implementation of this plugin is described in SignDocDefaultSigningHandler.

Use and configure a custom SigningEvent plugin

- Deploy the plugin as described in Plugin handling.
- Load, enable, assign and configure the plugin as described in Plugin administration.

Minimal SigningEvent implementation

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

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

This plugin 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.

Note 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

- 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 CIRRUS_HOME/plugins/default/classes
  - Jar files must be provided in CIRRUS_HOME/plugins/default/lib
  - See also Plugin handling
- Log in to the Administration Center or as an account administrator
  - Assign the full class name of this class, for example com.mycompany.plugins.MyRSASigner to the setting:
    System Settings > Plugins > 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 plugin and can be found in:

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

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

Core plugins

SignDoc default signing handler plugin

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 filed. 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

Notification plugin

Notification plugin description

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

The sample implementation uses a 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 plugin 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:

Parameter	Description
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 plugin 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:

Parameter	Description
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 plugins

SMS notification plugin

- Notification and the SMS plugin
- Registering an account with the SMS service
- Configuration

Notification and the SMS plugin

The notification service is designed to send a message to the user / signer. For a two 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 plugin. Thus additional delivery channels can be supported without changes to the product.

The core plugin supports an SMS notification channel.

Registering an account with the SMS service

To use the SMS plugin you will have to own a user account with Clickatell (http://www.clickatell.com).

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

- Accounts registered before November 2016 (Clickatell Central): use the NotificationSMSClickatell plugin.
- Accounts registered after November 2016 (Clickatell Platform): use the NotificationSMSClickatellPlatform plugin.

The plugins can be used at the same time in an installation. Usually one account will only use a single plugin. Both plugins 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 will describe the configuration for each type of plugin.

Clickatell Central API

Plugin load list

The load list specifies which plugins are supported by Cirrus. To make the SMS notification plugin usable, it has to be added to the load list.

The load list is a ‘,’ delimited list of plugin class names.

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

Enabling the plugin

Once the plugin has been loaded via the load list it can be enabled:

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

To enable the plugin, you have to set the setting

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

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

SMS plugin settings

Plugin settings are set as:

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

Example

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

Following settings are supported by the NotificationSMSClickatell plugin:

- **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, etc.) 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 plugin configured are described below:

1. Get access token for ksdadmin (no account). Use:
   users > create an authentication token
2. Set plugin load list account independent. Use:
   configuration > set configuration settings

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

3. Set plugin cfg (can be account specific). Use:
   configuration > set configuration settings

```
[
  {
    "k": "plugin.enabled.NotificationSMSClickatell",
    "v": "true"
  },
  {
    "k": "plugin.cfg.NotificationSMSClickatell.url",
    "v": "http://smscatcher.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

Plugin load list

The load list specifies which plugins are supported by Cirrus. To make the SMS notification plugin usable, it has to be added to the load list.

The load list is a ‘,’ delimited list of plugin class names.

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

Enabling the plugin

Once the plugin has been loaded via the load list it can be enabled:

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

To enable the plugin, you have to set the setting

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

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

SMS plugin settings

Plugin settings are set as:

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

Example

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

Following settings are supported by the NotificationSMSClickatellPlatform plugin:

- **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 plugin configured are described below:

1. Get access token for ksdadmin (no account). Use:
   users > authentication

2. Set plugin load list account independent. Use:
   configuration > set configuration settings

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

3. Set plugin 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 plugin

Package state change plugin description

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

The plugin 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 plugins that support the state change events.

Supported events

- **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 plugin intends to process the state change specified.

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

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

If additional information is needed by the plugin, 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:

Parameter	Description
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 plugin response as to if it intends to process that particular change or not:

Parameter	Description
OUT_SUPPORTED	The plugin 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 plugin. 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:

Parameter	Description
IN_ACCOUNT_OID	The account OID (string) of the signing package account.
IN_SIGNINGPACKAGE_OID	The signing package OID (string).
IN_SIGNINGPACKAGE_NAME	The signing package name (string).
IN_OLD_STATE	The signing package state before the state change (string).
IN_NEW_STATE	The new signing package state after the state change (string).
IN_OWNER_TOKEN	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:

Parameter	Description
IN_ACCOUNT_OID	The account OID (string) of the signing package account.
IN_SIGNINGPACKAGE_OID	The signing package OID (string).
IN_SIGNER_OID	The signer OID (string).
IN_SIGNER_FIRST_NAME	The signer first name, if recorded (string, optional).
IN_SIGNER_LAST_NAME	The signer last name, if recorded (string, optional).
IN_SIGNER_DISPLAY_NAME	The signer display name (usually first name + last name), if recorded (string, optional).
IN_OLD_STATE	The signer state before the state change (string).
IN_NEW_STATE	The new signer state after the state change (string).
IN_OWNER_TOKEN	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 plugins

KTA state change plugin

- KTA
- State change events
- Configuration

KTA

This plugin implements the communication with the KTA (Kofax TotalAgillity) system. KTA will create signing packages with documents to be signed as part of its workflow. After the documents have been signed, this plugin will report the status change back to the KTA system and let it continue its workflow.

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

Kofax SignDoc Standard introduces this new KTA state change plugin with version 2.1.0.

State change events

The plugin 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 plugin will generate KTA calls to inform KTA on the new processing state.

Configuration

Plugin load list

The load list specifies which plugins are supported by Cirrus. To make the KTA state change plugin usable, it has to be added to the load list.

The load list is a ‘,’ delimited list of plugin class names.

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

Enabling the plugin

Once the plugin 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 plugin, you have to set the setting

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

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

Plugin settings

Plugin settings are set as

plugin.cfg.<pluginId>.<settingName>

Example

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

Following settings are supported by the KTA state change notification plugin:

- **ktaurl** The KTA URL to send information to.
- **cirrusurl** The Cirrus (Signdoc Standard) REST API URL. The plugin 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 plugin is enabled it is required that the setting

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

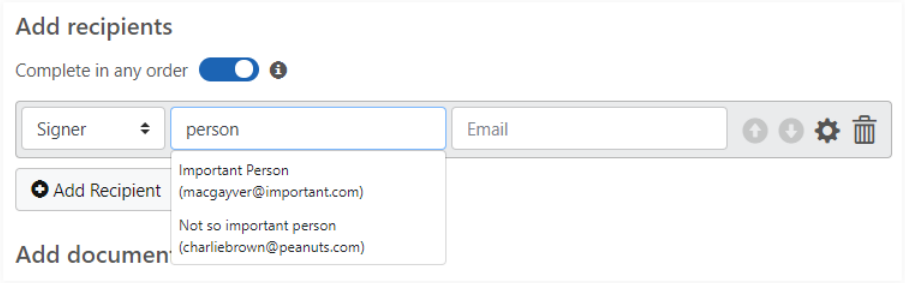
in the category "Documents and packages" is set.

Signer search plugin

Signer search plugin description

The signer search plugin enables you to write a plugin 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.



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:

Parameter	Description
searchname	The signer name for searching. A signer is included only if his name contains the provided text. This is case-insensitive. 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. 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 plugin.

The REST API implementation gets the signer list by calling the plugin 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 plugin implementation of SignerSearchEvent can be used for a signer search, it is necessary to define a specific pluginid for this event which is the SignDoc internal SignDocSignerSearch by default. Prerequisite for the usage of the plugin is that the specified plugin definition is included in the plugin.loadlist and it must be enabled, either globally or account specific.

Which plugin 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 plugin. The account independent default setting is SignDocSignerSearch for this configuration entry.

If no plugin is enabled for the SignerSearchEvent event then the "autocomplete" function is suspended because the REST call would produce always an empty list.

Supported events

- **Signer search event**

Signer search event

The following input parameters are provided in the SignerSearchEvent:

Parameter	Description
IN_USER_ID	The plugin 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 plugin 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 plugin implementation SignDocSignerSearch.

The following output is expected from the plugin for this event:

Parameter	Description
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 plugin 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.

Document scan plugin

Document scan plugin description

The document scan event lets you write a plugin 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

- **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 plugins registering this event and enabled for the account. If any of the plugins returns an invalid result, the document will be rejected.

The event input parameters are:

Parameter	Description
IN_CONTENT	The document content (binary, byte array), required
IN_NAME	The name of the file being uploaded (string), optional

The output parameters are:

Parameter	Description
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 plugins

ClamAV virus scan plugin

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 plugins on the affected account. All plugins can scan the document content. If any plugin responds with ‘true’ as to invalid content being detected, the document upload is rejected.

A sample implementation of this plugin 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:

- ClamAV is an open source virus scanner
- Is available free of charge
- Available on all major operating systems

Information on ClamAV can be found under https://www.clamav.net

Kofax 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 plugin a ClamAV server and a ClamAV REST endpoint need to be running.

- Documentation on running a ClamAV server can be found under https://www.clamav.net
- 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.

Kofax does not provide support on installing or running the ClamAV server. Refer to the links above and to https://www.docker.com for additional information.

Configure the ClamAV document scan plugin

Plugin load list

The load list specifies which plugins are supported by Cirrus. To make the ClamAV document scan plugin usable, it has to be added to the load list.

The load list is a ‘,’ delimited list of plugin class names.

```
plugin.loadlist = de.softpro.cirrus.plugins.document_scan.ScanClamAV
```

Enabling the plugin

Once the plugin has been loaded it can be enabled:

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

To enable the plugin you have to change the setting

```
plugin.enabled.ScanClamAV = true
```

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

Plugin settings

Plugin settings are set as

```
PLUGIN.CFG.<PLUGINID>.<SETTINGNAME>
```

Example

```
plugin.cfg.ScanClamAV.url
```

Following settings are supported by the ScanClamAV plugin:

- **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 one can upload the test EICAR virus signature as a document and verify that the scanner works (http://www.eicar.org).

TSP plugin

Trusted service provider plugin 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.

This interface is used both by SignDoc Standard and SignDoc Web, which accounts for some peculiarities with the configuration. In SignDoc Standard the IConfigurablePlugin interface and the account-specific instantiation is used to inject account-specific configuration data via IPlugin. SignDoc Web does not support account-specific configuration. Therefore, the event interfaces will use an optional settings parameter that SignDoc Web will pass with every call and that will override general settings. Thus, if a settings parameter is present, it should be given precedence over any settings injected via IPlugin at plugin instantiation.

Supported events

The trusted service provider interface supports three events:

- **TSP info event**
  Provides information about the TSP provider the plugin 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 plugin supports:

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

Parameter	Description
IN_SETTINGS	Optional. If present, the settings map will completely override the settings provided at plugin instantiation via IPlugin.injectPluginConfiguration. This is necessary, because SignDoc Web has no account-specific plugin mechanism, thus the account-specific parameters need to be passed for each call. The plugin has to be able to re-initialize on a ‘by call’ basis.

The output will provide provider-specific information needed to display input and information pages related to the TSP validation and signing process:

Parameter	Description
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 plugin.

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 plugin 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:

Parameter	Description
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 plugin instantiation via *IPlugin.injectPluginConfiguration*. This is necessary, because SignDoc Web has no account-specific plugin mechanism, thus the account-specific parameters need to be passed for each call. The plugin has to be able to re-initialize on a 'by call' basis.
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:

Parameter	Description
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:

Parameter	Description
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.

Parameter	Description
IN_SETTINGS	Optional. If present, the settings map will completely override the settings provided at plugin instantiation via IPlugin.injectPluginConfiguration. This is necessary, since SignDoc Web has no account-specific plugin mechanism, thus the account-specific parameters need to be passed for each call. The plugin has to be able to re-initialize on a ‘by call’ basis.
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 plugin. **Note** 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 plugin.

The output data includes:

Parameter	Description
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:

Parameter	Description
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 plugin instantiation via IPlugin.injectPluginConfiguration. This is necessary, because SignDoc Web has no account-specific plugin mechanism, thus the account-specific parameters need to be passed for each call. The plugin has to be able to re-initialize on a ‘by call’ basis.
IN_SIGNATURE_PROCESS_TOKEN	The signature token that identifies this signing process (received by PostDocumentSignatureEvent).

The output data includes:

Parameter	Description
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.

Sample development package

A complete sample development package for TSP plugins can be found in

```
INSTALLDIR\signdoc_home\interfaces\plugins\simpletsp-<version>.zip
```

Note The sample development package can be used for platform independent development (Windows, Linux), but the contained service installer can only be run on a Windows operating system.

Core plugins

TSP plugin

- Digital document signing and trusted service providers
- Registering an account with Bundesdruckerei
- Configuration
- New SignDoc 3.0.0 TSP features

Digital document signing and trusted service providers

As of release 1.3.1 Kofax SignDoc supports digitally signing documents according to the EIDAS standard. The document is signed by an external trusted service provider, independent of the Kofax SignDoc installation. The TSP is accessed via a plugin interface, allowing for multiple TSP support and expansion independent of a new product release.

The standard plugin delivered with the product supports DTrust GmbH / Bundesdruckerei GmbH as a trusted service provider. Other providers can be added by writing and configuring additional plugins. This chapter describes the configuration of the plugin mentioned above.

Registering an account with Bundesdruckerei

To use the service, both the operator and the signers need to have an account registered with Bundesdruckerei GmbH.

The operator has to register with Bundesdruckerei GmbH and obtain a partner id and authentication settings.

Any signer that will have a digital signature appended during the signing process has to register with Bundesdruckerei GmbH as a user.

Click https://cloud-ref.sign-me.de/signature/start to go to the registration webpage of Bundesdruckerei GmbH.

Configuration

All settings described here are configured via the Cirrus configuration service. No specific SignDoc Web configuration is needed, since the component will request the configuration settings via the Cirrus configuration API.

Plugin load list

```
plugin.loadlist = de.softpro.cirrus.plugins.tsp.TSPBundesdruckerei
```

The load list specifies which plugins are supported by Cirrus. This is a system-wide setting and can only be set account independent. To load the TSP plugin it has to be added to it, which is a list of ‘,’ delimited class names.

Enabling the plugin

Once a plugin has been loaded via the load list it can be enabled:

for one or more specific accounts

for all accounts globally

To enable the plugin, you have to set the setting

```
plugin.enabled.TSPBundesdruckerei = true
```

either as a global setting (no account) or for one or more specific account ids.

TSP plugin settings

Plugin settings are set as

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

Example

```
plugin.cfg.TSPBundesdruckerei.visibility
```

They can be set either as global values or on an account-specific basis.

Following settings are supported by the TSPBundesdruckerei plugin:

- **regurl** The registration URL where new Bundesdruckerei GmbH users can register an account.
- **apiurl** SignMe API url.
- **basicauthuid** Basic authenticaton user id received from Bundesdruckerei GmbH.
- **basicauthpw** Basic authentication password received from Bundesdruckerei GmbH.
- **partnerauthuid** Partner authentication user id received from Bundesdruckerei GmbH.
- **partnerauthpw** Partner authentication password received from Bundesdruckerei GmbH.
- **visibility** Visibility of document signatures. Possible values are INVISIBLE, FIRSTPAGE, ALLPAGES, default being INVISIBLE.
- **padesform** PAdES format. Possible values are BASIC or ENHANCED, default being BASIC. When using ENHANCED the parameter visibility has to be set to FIRSTPAGE or ALLPAGES.
- **replacefilename** Replace file names sent for signing if they contain unsupported characters. Possible values: true or false. If true the file name will be changed for display in SignMe to “document.pdf”.

Sample TSP plugin configuration

Enable TSP plugin for Bundesdruckerei via configuration REST API (required role SUPERUSER):

```
POST /rest/v8/configuration
```

Add plugin to plugin load list:

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

Enable plugin for specific account (here signdoc):

```
POST /rest/v7/configuration?accountid=signdoc
```

```
{
  "list": [
   {
    "k": "plugin.enabled.TSPBundesdruckerei",
    "v": "true"
    }
  ]
}
```

Provide plugin settings for specific account:

```
{
  "list": [
    {
      "k": "plugin.cfg.TSPBundesdruckerei.basicauthuid",
      "v": "basic"
    },
    {
      "k": "plugin.cfg.TSPBundesdruckerei.basicauthpw",
      "v": "kjusd-hgd./Jgd$3!"
    },
    {
      "k": "plugin.cfg.TSPBundesdruckerei.partnerauthuid",
      "v": "partner"
    },
    {
      "k": "plugin.cfg.TSPBundesdruckerei.partnerauthpw",
      "v": "lnjlkjsf/%fgLNf=FnmXCkflfkg"
    },
    {
      "k": "plugin.cfg.TSPBundesdruckerei.apiurl",
      "v": "https://cloud-ref-sp.sign-me.de:443/api/v2"
    },
    {
      "k": "plugin.cfg.TSPBundesdruckerei.padesform",
      "v": "ADVANCED"
    },
    {
      "k": "plugin.cfg.TSPBundesdruckerei.visibility",
      "v": "INVISIBLE"
    },
```

```
    {
      "k": "plugin.cfg.TSPBundesdruckerei.regurl",
      "v": "https://cloud-ref.sign-me.de/signature/start"
    }
  ]
}
```

New SignDoc 3.0.0 TSP features

As part of release 3.0.0, a TSP plugin 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 plugin.

Note 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 plugin.

TSP-Loopback-Plugin is a test TSP plugin which can be used for reference purposes.

Activating the TSP-Loopback-Plugin:

Add ‘de.softpro.cirrus.plugins.tsp.TSPLoopbackPlugin‘ to the existing ‘plugin.loadlist’ configuration from the administrator's section. This plugin 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’ plugin should use below configurations:

- plugin.cfg.<pluginId>.tsp.signature.enabled
  Defines if the TSP plugin supports 'TSP' signing mode.
  Default value: true

  Note It is necessary that the plugin if defines this configuration should have a default value ‘true’ and have plugin setting type ‘BOOLEAN’. The ‘clientCertificateRequired’ option for a signer is mutually exclusive when used along with this configuration i.e. a signer with ‘clientCertificateRequired’ set to true will not be able to have a signature field with ‘TSP’ signing mode and vise 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 below configurations 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.

  Note If this configuration is not provided a redirect message will be displayed by default to the signer. It should have plugin 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

  Note This configuration should support MediaType.APPLICATION_OCTET_STREAM and have plugin 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.

  Note If this configuration is not provided, a redirect message will be displayed by default to the signer. It should have plugin 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 plugin 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 plugin.

- plugin.cfg.<pluginId>.tsp.document.enabled
  Defines if the TSP plugin supports document signing using 'TSP'.

  Note It is necessary that the plugin if defines this configuration should have a default value ‘true’ and have plugin 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

Mail

SMTP configuration

Description

All possible SMTP related settings can be configured per SignDoc account or as a global database configuration using the managed 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. '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 managed 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 SignDocBankIdService-3.0-windows.zip to any folder.
2. Configure service in file SignDocBankId.xml (tailored as per above details).
3. Run service_up.cmd.
4. Test if service is started by calling localhost:6614 in the Browser. You should see now SignDoc BankID 3.0 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 managed client:

- **cirrus.security.external.authentication.name**
  E.g. 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, e.g. http://localhost:6614/extauth.

The package creator should then assign this external authentication to the signer in the signer's settings while creating the signer and adding 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, user needs to have a BankID application installed on the desktop he is performing the authentication. The user does not have to start the application manually for this to work. When 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**
  User 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 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 clicking the **Cancel** button 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 the **Cancel** button, 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 below steps:

Sign in to Kofax SignDoc Administration Center.

Go to System Settings > Plugins.

1. Under General add 'de.softpro.cirrus.plugins.tsp.TSPBankID' to the configuration plugin.loadlist.
2. Under Enabled set plugin.enabled.TSPBankID to true.
3. Under Configuration, you can see TSPBankID plugin. When you click on the same you can modify the TSPBankID plugin related configurations.

The configuration under point 1. can only be handled from the Administration Center, the others can be managed from other Admins 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**
  User 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 the **Cancel** button 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 on the cancel button, 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 information on BankID, see the following links:

https://www.bankid.com/utvecklare/rp-info

https://www.bankid.com/assets/bankid/rp/bankid-relying-party-guidelines-v3.5.pdf

Chapter 7

Tenant-specific URL

Important SignDoc Standard before version 2.1.0 was mainly configured with the configuration file cirrus.properties. This file moved to `INSTALLDIR\_conf_templates\cirrus.properties` with version 2.1.0.

Since SignDoc Standard 2.1.0, it is highly recommended to use the file `INSTALLDIR\service_configuration.properties` (instead of cirrus.properties) whenever it is required to configure SignDoc with a configuration file. Configurations set in this file are applied as Java system property and have therefore highest precedence.

It is possible to create several accounts in SignDoc Standard for different tenants.

If you have several accounts you have to specify with which account you want to login for processing signing packages. This can be achieved by entering an accountid in the login panel.

As a tenant user it is very uncomfortable to enter an accountid each time for login.

A possibility is implemented to select in advance an account via tenant-specific URL.

Let’s assume that the SignDoc Standard application can be reached via https://www.signdoc.com:8083/cirrus.

We have two accounts in SignDoc Standard, one for customer "Customer One" and one for customer "Customer Two".

The administrator can configure in DNS (or for testing also in the local hosts file) that the SignDoc Standard server can be reached also via

- cone.signdoc.com for tenant "Customer One"

and

- ctwo.signdoc.com for the customer "Customer Two"

The domain name prefix cone and ctwo has to be configured in the SignDoc Standard application for identifying the appropriate account.

This can be done by the Server Administrator in the Account Details dialog using the field dnslabel.

For the tenant "Customer One" he has to enter cone in the entry field dnslabel and ctwo for tenant "Customer Two".

Important To activate this feature the administrator has to enter cirrus.tenant.url.supported=true in %CIRRUS_HOME%/conf/cirrus.properties. It is not activated by default! The DNS label must be unique for each tenant and at most 63 characters.

Implementation details

The SignDoc Standard application parses the domain name in the (login) URL and extracts the tenant-specific part to provide a request parameter dnslabel with this value.

Example

If the user calls URL cone.signdoc.com the application adds a request parameter dnslabel=cone to the call (http://cone.signdoc.com will be mapped to http://signdoc.com/cirrus?dnslabel=cone).

This dnslabel has to match with a dnslabel attribute in the ACCOUNT table for identifying an account.

In this case the SignDoc Standard application knows the requested account and the entry field for the accountid is omitted in the login dialog.

Chapter 8

Google Chrome Group Policy (GPO)

When using the Google Chrome Group Policy (GPO) some adjustments have to be made to support our 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 "
```