Kofax SignDoc Standard

Administrator's Guide

Version: 2.2.1

Date: 2019-11-18

© 2019 Kofax. All rights reserved.

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

Table of Contents

Preface..........5

Training..........5

Related documentation..........5

Getting help with Kofax products..........5

Definitions..........6

Chapter 1: Licensing..........7

License handling..........7

SignDoc license..........7

Account license..........8

Global license..........8

Reset license (special license)..........9

Chapter 2: Logging..........10

On-premise logging..........10

Request logging..........11

Chapter 3: Configuration..........13

Configuration service..........13

Configuration levels..........14

Change configuration options..........14

Configuration files..........18

Configure the 'autoprepare' functionality..........19

Fonts used in the final document..........19

Configuration values..........20

System..........20

Documents and packages..........22

Security..........23

Mail..........24

Plugins..........28

Client..........28

Advanced signing settings..........31

Chapter 4: Plugins..........33

Plugin handling..........33

Plugin administration..........35

Plugin development..........41

Plugin interface..........41

Plugin implementation....42
How SignDoc uses plugins.... 42
Signing plugin.... 43
SigningEvent plugin description.... 43
Minimal SigningEvent implementation....43
SigningRSA interface....46
Minimal SigningRSA implementation.... 47
Core plugins....50
Notification plugin.... 51
Notification plugin description....51
Core plugins....52
Package state change plugin....56
Package state change plugin description....56
Core plugins....58
Signer search plugin....60
Signer search plugin description.... 60
Document scan plugin....62
Document scan plugin description.... 62
Core plugins....62
TSP plugin.... 65
Trusted service provider plugin description....65
Core plugins....68
Chapter 5: Monitoring....72
Overview.... 72
Monitoring the KSD cluster.... 73
Monitoring protocol.... 73
Metrics being collected....74
Monitoring server.... 74
SignDoc Standard application metrics.... 74
Metric overview....75
Configuration....76
Metric description....76
Example setups.... 80
Monitoring service sample setup....80
Collectd sample setup for collecting system metrics.... 81
Glossary.... 83
Chapter 6: Tenant-specific URL....85
Chapter 7: Google Chrome Group Policy (GPO)....87

Preface

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

Training

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

Related documentation

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

https://docshield.kofax.com/Portal/Products/en_US/SD/2.2.1-kjbcp1n42d/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*

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
The directory with the unpacked signdoc-standard-2.2.1.zip file. See Related documentation, *SignDoc Standard Installation Guide*, chapter "Quickstart procedure".

CIRRUS_HOME, SDWEB_HOME
The directories of the web applications that compose SignDoc Standard. 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 parameter:

`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.

Note 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 internal logging framework XjLog which uses Logback (http://logback.qos.ch/). Logback is intended as a successor to the popular log4j project.

By default SignDoc Standard creates log files on disk. Cirrus-specific log files are written to %CIRRUS_HOME%/logs.

%CIRRUS_HOME% is an environment variable which has to point to a valid directory. XjLog can be configured via XjLog.xml file which is available in %CIRRUS_HOME%/conf.

Cirrus writes its log by default into a file cirrus.log. The RollingFileAppender which is used for logging here uses FixedWindowRollingPolicy as rollingPolicy and SizeBasedTriggeringPolicy as triggeringPolicy. SizeBasedTriggeringPolicy looks at the size of the currently active file. If it grows larger than the specified size, it will signal the owning RollingFileAppender to trigger the rollover of the existing active file. With FixedWindowRollingPolicy cirrus.log is renamed to cirrus1.log if the maximum log file (currently 10MB) is reached. A new cirrus.log file is then created where new log entries are written. In the second rollover cirrus1.log is renamed as cirrus2.log and cirrus.log is renamed as cirrus1.log. The upper bound index is 10 (by default). If this index is reached this (oldest) log file cirrus10.log will be deleted and the previous (cirrus9.log) is renamed to cirrus10.log.

For better identifying the error logs are written to a separate subdirectory

```
%CIRRUS_HOME%/logs/error
```

The errors of each session are logged into a separate error log. The file name is either the current sessionid or the current timestamp if the error is independent from a specific session.

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

Request logging

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.

Each http request to SignDoc Standard and the appropriate response is logged.

For example, the browser call

```
http://beaker:8081/cirrus/
```

to the SignDoc Welcome page could be logged as follows in cirrus.log:

```
I BEAKER 470D889C58E7718E3BC8822CA78FAC5F 2015-11-26 14:19:16.062 [http-
bio-8081-exec-2] de.softpro.cirrus.web.helper.RequestLogFilter - REQUEST: POST
http://beaker:8081/cirrus/WEB-INF/jsp/script/relogin.jsp

I BEAKER C4E2753D8977EF1D0C4A93C8C32C7BCD 2015-11-26 14:19:16.100 [http-
bio-8081-exec-2] de.softpro.cirrus.web.helper.RequestLogFilter - RESPONSE
Status Code: 200 (for [POST] request URI: /cirrus/WEB-INF/jsp/script/
relogin.jsp) in 38ms

I BEAKER C4E2753D8977EF1D0C4A93C8C32C7BCD 2015-11-26 14:19:16.106 [http-
bio-8081-exec-2] de.softpro.cirrus.web.helper.RequestLogFilter - REQUEST: GET
http://beaker:8081/cirrus/user$login.re?user=1401 => Parameter: [user=1401]

I BEAKER C4E2753D8977EF1D0C4A93C8C32C7BCD 2015-11-26 14:19:16.189 [http-
bio-8081-exec-2] de.softpro.cirrus.web.helper.RequestLogFilter - RESPONSE
Status Code: 200 (for [GET] request URI: /cirrus/user$login.re) in 83ms
```

In the configuration file

```
%CIRRUS_HOME%/conf/cirrus.properties
```

the following settings can be set or changed for getting less or more detailed request and response information:

- Log all requests including parameters.
  cirrus.logging.all_requests=false
- Log all servlet requests including parameters.
  cirrus.logging.servlet_requests=true
- Log all REST requests including parameters.
  cirrus.logging.rest_requests=true

- Log all cookies.
  cirrus.logging.request.all_cookies=false
- Log all JSESSIONID cookies.
  cirrus.logging.request.jsession_cookies=false
- Log all request headers.
  cirrus.logging.request_headers=true
- Log also body (if exists) of request (if the request is enabled for logging).
  cirrus.logging.request_body=false
- Output limit of the requested body log entry (in bytes), set -1 for unlimited.
  cirrus.logging.request_body_length=200
- Log all response headers.
  cirrus.logging.response_headers=true
- Log response content (if available).
  cirrus.logging.response_content=false
- Output limit of the response content log entry (in bytes), set -1 for unlimited.
  cirrus.logging.response_content_length=200
- Include specific URIs (or parts from it) for request and response logging
  cirrus.logging.include_uri_list_request=

  Example

  cirrus.logging.include_uri_list_request=/LogService/,/ConfigurationService
- Exclude specific URIs (or parts from it) from request and response logging
  cirrus.logging.exclude_uri_list_request=

  Example

  cirrus.logging.exclude_uri_list_request=/LogService,/ConfigurationService
- Include specific URIs (or parts from it) for response content logging
  cirrus.logging.include_uri_list_response_content=

  Example

  cirrus.logging.include_uri_list_response_content=/services/,/rest/]
- Exclude specific URIs (or parts from it) from response content logging.
  cirrus.logging.exclude_uri_list_response_content=
- Exclude specific requested resource files (file extension which must be at the end of the URI) from request and response logging.
  cirrus.logging.exclude_uri_list_file_suffix=.gif,.png,.jpg,.css,.js,.svg,.ico

Prerequisite

The log level of RequestLogFilter must be set to INFO in %CIRRUS_HOME%/conf/XjLog.xml (default):

```
<logger name="de.softpro.cirrus.web.helper.RequestLogFilter" level="debug"
 additivity="false">
<appender-ref ref="CIRRUS" />
<appender-ref ref="REDIS" />
</logger>
```

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.

Starting with the release 1.3, some configuration options have been moved from configuration files to the configuration service.

Monitoring and the metrics processed can be configured in cirrus.properties configuration file with the following properties:

- **monitoring.host** The hostname or IP address of the monitoring server the metrics are sent to. If left empty, no monitoring data is sent (default).
- **monitoring.port** The port number where monitoring data is sent to. Defaults to 2003
- **monitoring.protocol** The protocol being used (TCP or UDP). Defaults to UDP.
- **monitoring.filter.include** Regular expression to specify which metrics form the available list should be reported. Defaults to all.
- **monitoring.filter.exclude** Regular expression specifying which metrics should be excluded form monitoring. By default, following statistic information is being excluded: count, m5_rate, m15_rate, max, min, mean_rate, p50, p75, p95, p98, p99, p999, stddev

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:

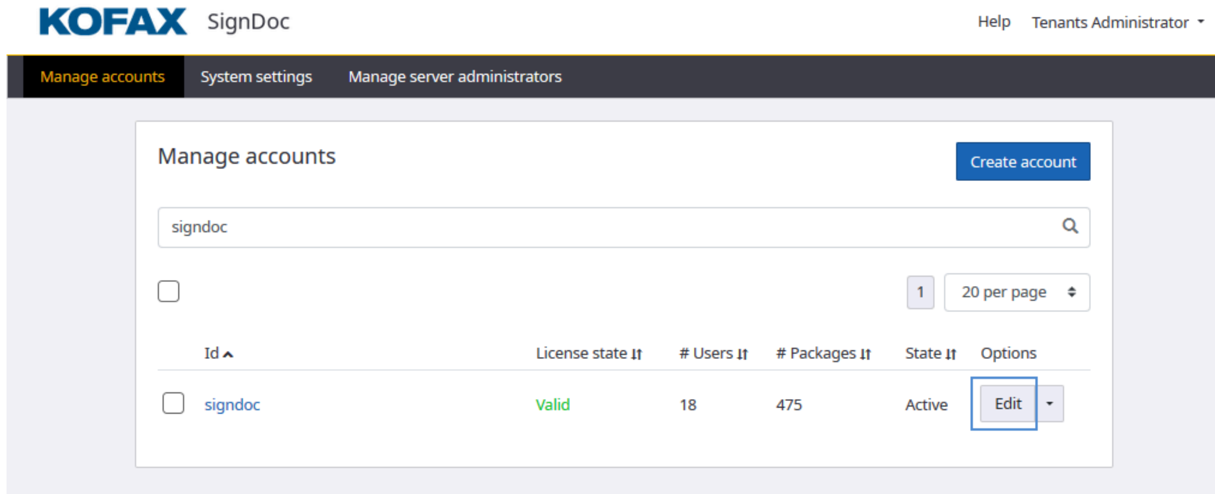
- A title
- The configuration option id
- A description of the configuration option

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.

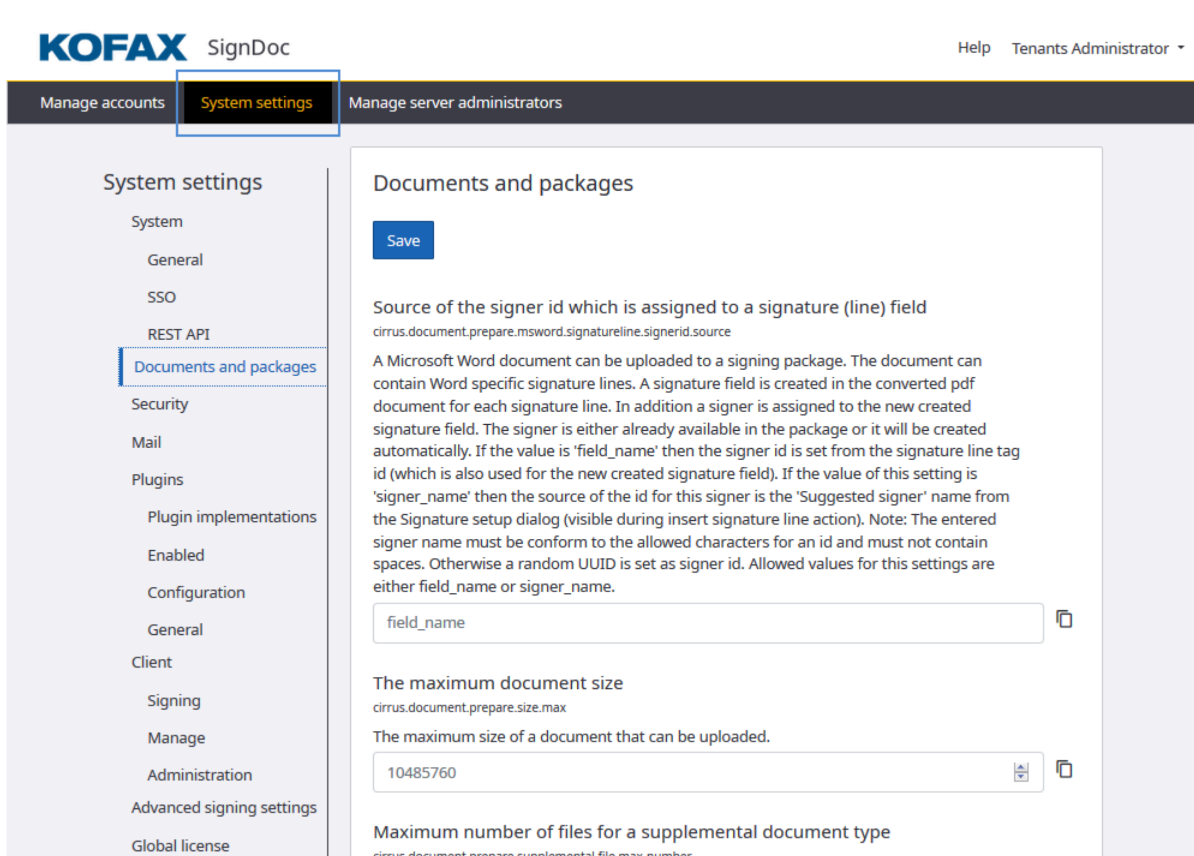
A configuration value will be validated by the client before it is used.

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:

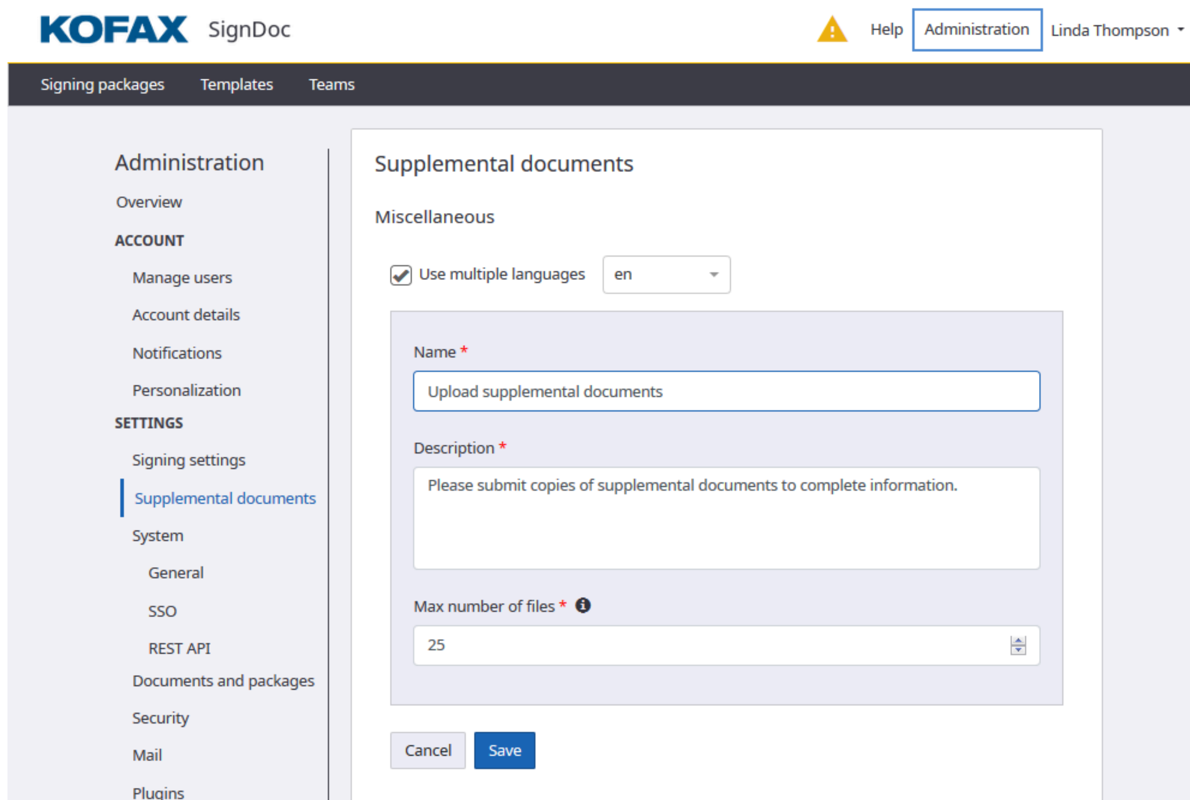


Clicking **Edit** will take you to a similar view as above, where the account-specific settings can be edited.



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.

Kofax SignDoc REST API Documentation V7

accounts : Dealing with accounts. Show/Hide | List Operations | Expand Operations

configuration : Dealing with configuration settings. Show/Hide | List Operations | Expand Operations

GET /rest/v7/configuration Get configuration settings

GET /rest/v7/configuration/descriptions Get configuration setting descriptions

POST /rest/v7/configuration Set configuration settings

DELETE /rest/v7/configuration Delete configuration settings

documents : Dealing with documents. Show/Hide | List Operations | Expand Operations

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.esign.consent** This property defines the default e-sign consent text that is applied to every new account and can be changed later in the account administration section of the application. Default: IMPORTANT! This is an example E-Sign consent text. Replace this text with your own legal text in your account preferences according to your local laws: This E-Sign Consent ("Consent") applies to all Communications for services and accounts offered or accessible through Kofax SignDoc that are not otherwise governed by the terms and conditions of an electronic disclosure and consent. By selecting the "I agree" acknowledgment on this page, you agree to sign documents electronically and understand that the provision of such signatures constitutes a legally binding transaction according to the applicable local E-Sign legislature.
- **cirrus.fail.login.max.attempts** This property is used to determine after how many failed login attempts a user will be suspended. When a user is suspended due to reaching the maximum configured number of login attempts, an email will be send to that user. When a successful login is made, the user-specific number of failed logins is reset. Default: 10
- **cirrus.finaldocument.font.name** With this property the font used in the final document is set. Default: DejaVu Sans
- **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.

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. Copy any fonts that might be used in the application in that location.

A preferred font can be specified by changing the value of the cirrus.finaldocument.font.name property. Add the configuration properties to `INSTALLDIR\service_configuration.properties`.

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.
- **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, SigDoc tries to authenticate the user automatically by querying the configured Single Sign-on authentication module. Default value: No
- **Create new users**
  cirrus.sso.create.user
  Automatically create new Single Sign-on users in SignDoc. Default value: Yes
- **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: Yes

REST API settings

- **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 (needed by the Manage Client).
- **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.
- **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.
- **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.
- **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.
- **Require password to change a user's email address**

  cirrus.rest.user.email_change.require_password

  If enabled (i.e. set to true) the authenticated user must also supply the own password to change any user's email address.

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.
- **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 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.
- **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 usable it will be rejected.
- **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.

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).

Mail

This section lists the mail related 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.
- **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.
- **Final document notification**

  cirrus.mail.finaldocument.notification.enabled

  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.
- **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.

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.

- **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.
- **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 disabled, the configured link for the online help will not be shown in the GUI. See also: client.signing.general.onlinehelp.url
- **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.
- **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.
- **Show footer**

  client.signing.view.general.footer.visible

  Defines if the footer is shown in the signing client.
- **Show header**

  client.signing.view.general.header.visible

  Defines if the header is shown in the signing client.

- **Show instructions**
  client.signing.view.general.instructions.visible
  Defines if the instructions are shown in the signing client.
- **Show wizard-steps**
  client.signing.view.general.wizardsteps.visible
  Defines if the wizard-steps are shown in the signing client.
- **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 false the in-person signing view is skipped when there is only one signer in an in-person signing session.
- **Show download in "Review & Sign" view**
  client.signing.view.reviewsign.download.visible
  Defines if the download actions are shown in the review & sign view.
- **Show progress in "Review & Sign" view**
  client.signing.view.reviewsign.progress.visible
  Defines if the progress bar is shown in the review & sign view.
- **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.
- **Show "Review & Sign" view**
  client.signing.view.reviewsign.visible
  Defines if the review & sign view is shown in the signing client. If set to false 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.
- **Show "Welcome" view**
  client.signing.view.welcome.visible
  Defines if the welcome view is shown in the signing client. If set to false the welcome view is skipped when no signer authentication (access code or external authentication) is used.

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.
- **Skip the landing page**
  client.general.skip.landing
  Skips the initial landing page displayed before the login form. Default value: No
- **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 disabled, the configured link for the online help will not be shown in the GUI. See also: client.manage.general.onlinehelp.url
- **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 true, recipients cannot be entered manually, but must be selected from a list.
- **Takeover attributes of selected signer**

  client.manage.signer.autocomplete.takeover

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

Administration Client related settings

- **Admin Client online help URL**

  client.admin.general.onlinehelp.url

  The URL for the Admin Client online help. See also: client.admin.general.onlinehelp.visible
- **Enable Admin Client online help**

  client.admin.general.onlinehelp.visible

  If disabled, the configured link for the online help will not be shown in the GUI. See also: client.admin.general.onlinehelp.url

Advanced signing settings

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

- **2FA access code length**

  cirrus.security.2fa.accesscode.length

  The length of the generated random access code for the 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.
- **Signature device selection priority when signing**

  client.signing.device.priority

  Defines the signature device selection priority (i.e. the SignWare padclass attribute) when signing. (SPTabletWSignPad, SPTabletWTablet, SPTabletWSignPad;SPTabletWTablet, SPTabletWTablet;SPTabletWSignPad) Values must be entered without whitespaces.
- **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).

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 enables 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 special: 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.

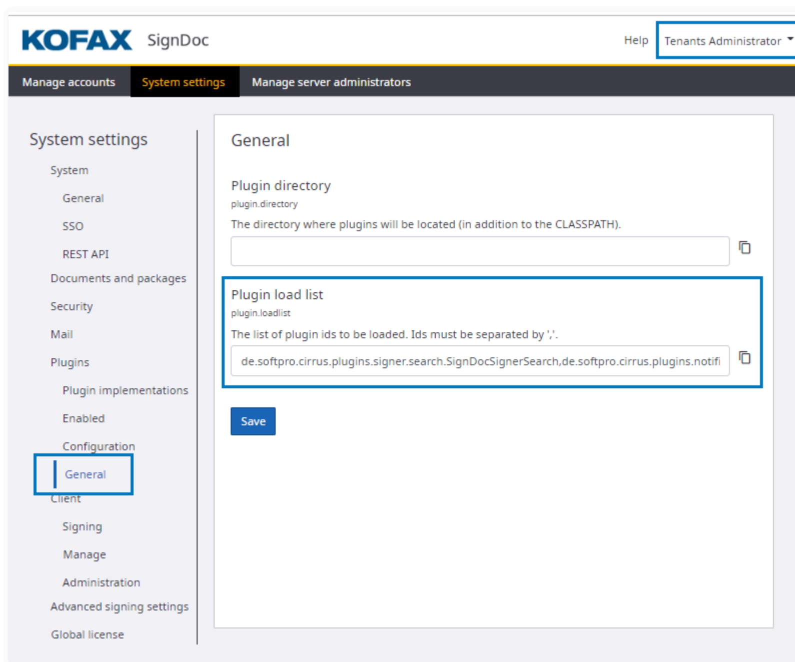
Tasks that can be done as account-specific administration are:

- 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**.

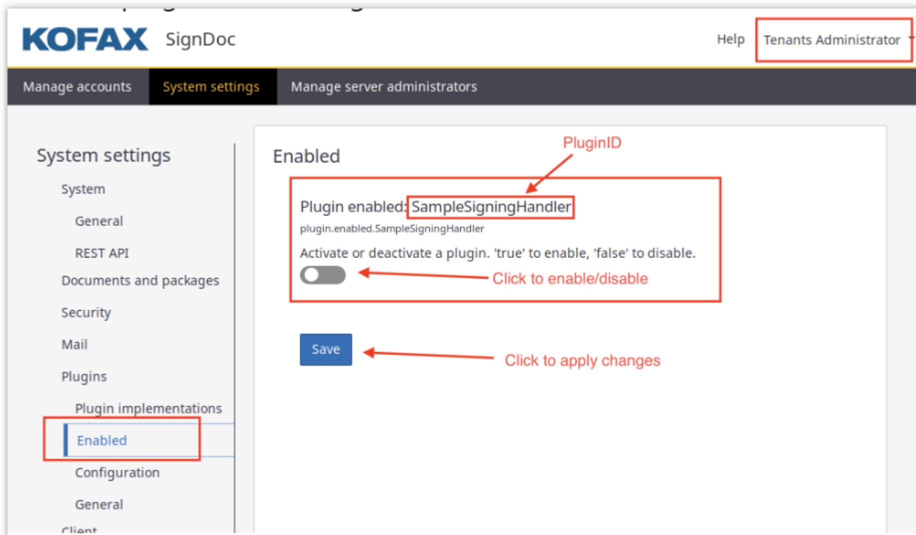
Add the class name of the plugin to the **Plugin load list** and save the settings.



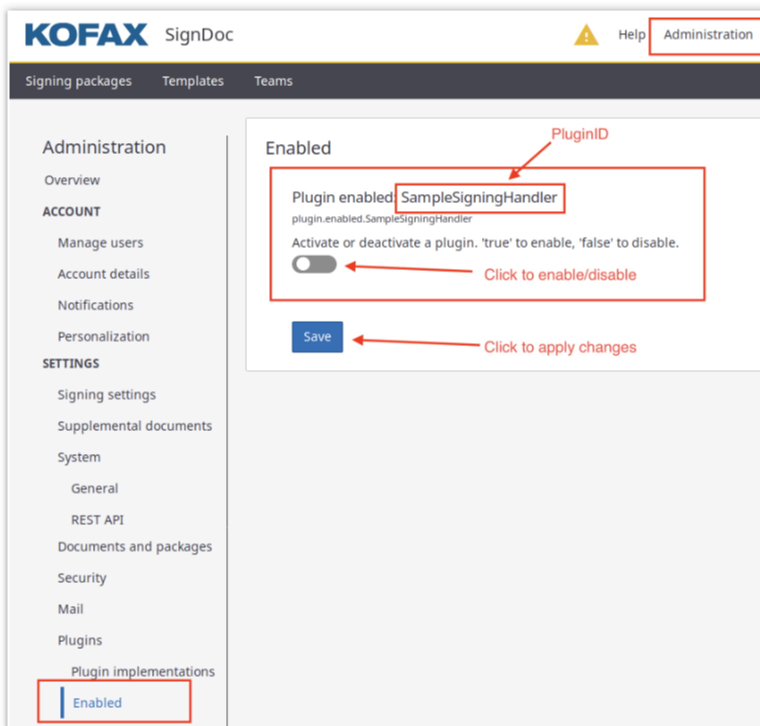
Enable or disable a plugin (Administration Center and account administration in Manage Client)

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.

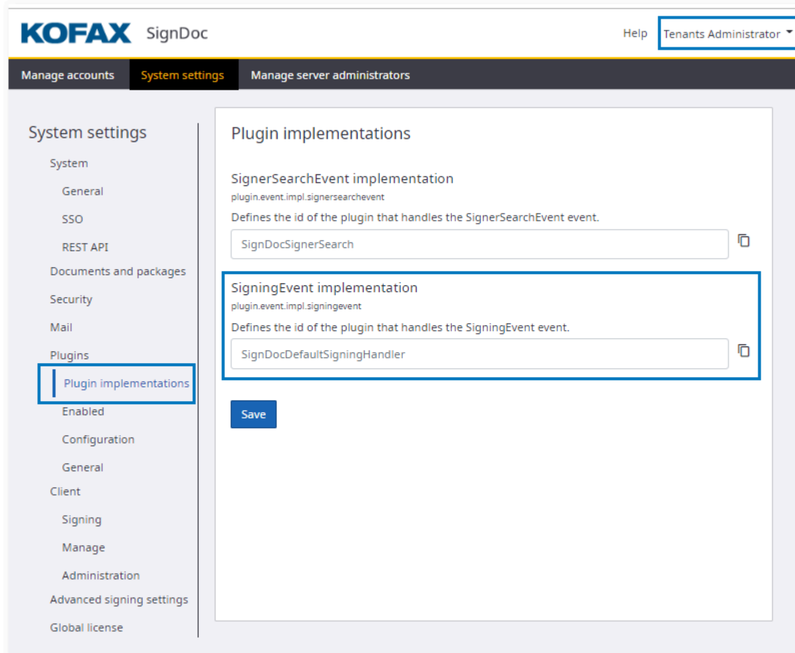


Assign a plugin implementation to a specific event

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

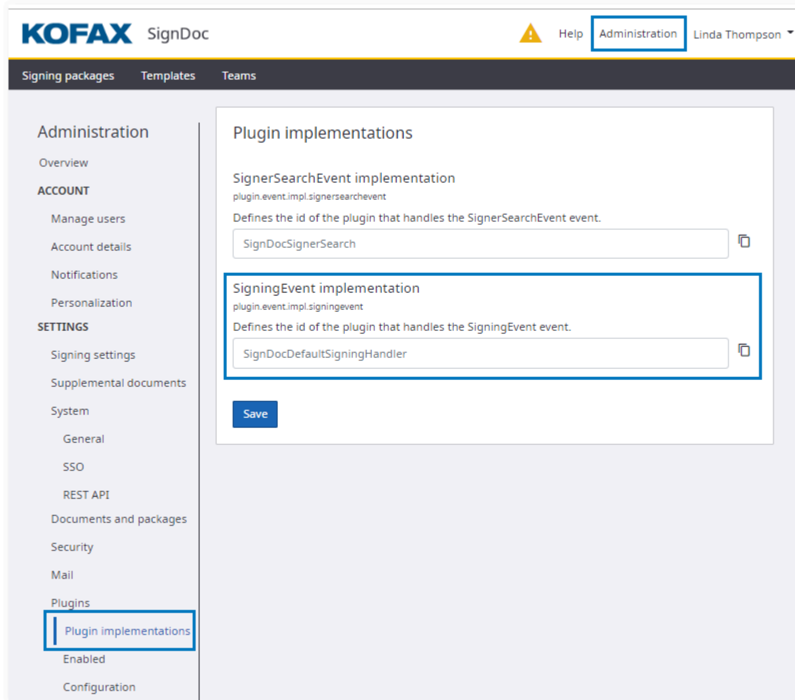
In the Administration Center, on the **System Settings** menu, click **Plugins** > **Plugin implementations**.

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.



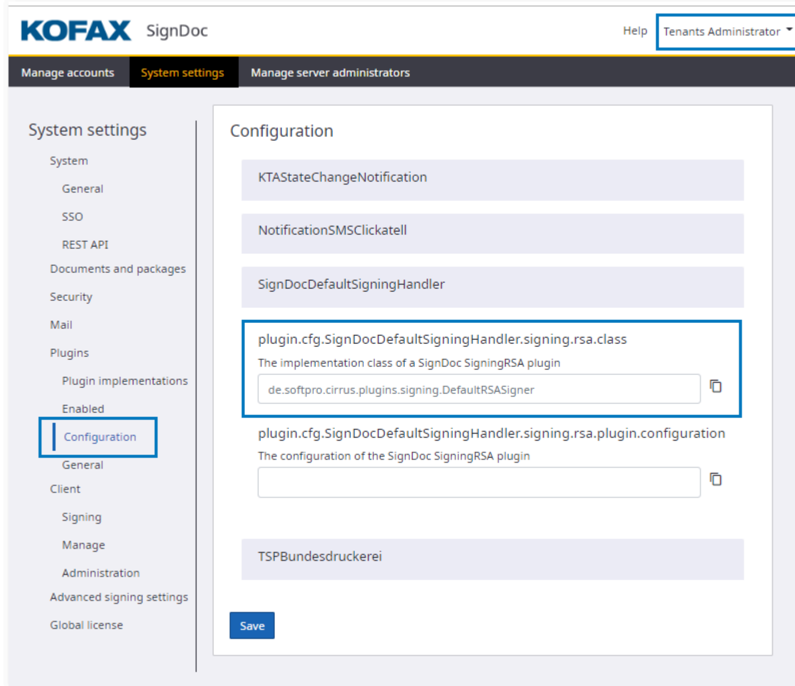
After a plugin has been enabled, it must be assigned to a specific event to modify or extend the SignDoc Standard behavior.

Configure plugins

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

Select a plugin id from the list.

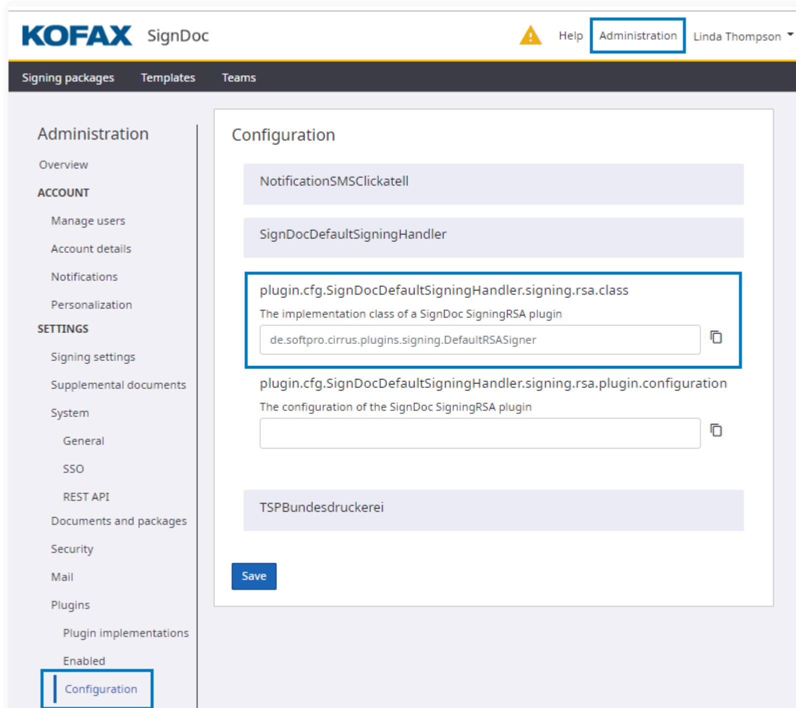
Set the value for the implementation class and save the settings.



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 SignDocDefaultSignerHandler.

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

This is a minimal implementation of a SigningEvent plugin. This plugin uses SignDoc SDK to sign the document's signature field.

The complete example can be found in

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

Excerpt of an example file:

```
import static de.softpro.pdflib.SignDocPDF.INIT_PARAM_SDSDK_ROOT_CERT;

/**
 * This class is a skeleton to implement an completely customized signing method.
 * It default implementation signs a signature field using SignDoc SDK.
 * To use the plugin it is required to...
 * 1) Add this class (com.mycompany.plugins..MinimalSigningEventImplementation)
 *    in Administration Center or via REST API to the plugin.loadlist setting
 *    System Settings -> Plugins -> General -> plugin.loadlist
 * 2) To activate the plugin in Administration Center
 *    System Settings -> Plugins -> Enabled -> MinimalSigningEventImplementation
 * 3) To assign this class PLUGINID (MinimalSigningEventImplementation) to the setting
 *    System Settings -> Plugins -> Plugin implementations ->
 plugin.event.impl.signingevent
```

```
 */
public class MinimalSigningEventImplementation extends AbstractKofaxPlugin implements
 IConfigurablePlugin {

    private static final XjLog log =
 XjLog.getLogger(MinimalSigningEventImplementation.class);

    private static final String PLUGINID =
 MinimalSigningEventImplementation.class.getSimpleName();

    private final Map<String, Object> CONFIG_MAP = new ConcurrentHashMap<>();

    // 2 simple config options
    private static final PluginSettingDescription config_option_1 =
            new PluginSettingDescription("config_option_1", PluginSettingType.STRING,
 "Description of config_option_1", ".+", false, "false");
    private static final PluginSettingDescription config_option_2 =
            new PluginSettingDescription("config_option_2", PluginSettingType.STRING,
 "Description of config_option_2", ".+", false, "false");

    @Override
    public PluginSettingDescription[] getSettingDescriptions(Locale locale) {
        return new PluginSettingDescription[] {config_option_1, config_option_2};
    }

    @Override
    public IEventResult eventCallback(IPluginEvent event, Map<String, Object> params)
 throws PluginException {
        try {
            if (event == null) {
                log.info("No event data received");
                throw new PluginException(EPluginMsgs.PLUGIN_UNSUPPORTED_EVENT,
 getPluginId(), "null");
            } else if ((new
 SigningEvent()).getEventName().equals(event.getEventName())) {
                // get the default plugin signing parameters
                PluginSigningParameters pluginSigningParameters =
 (PluginSigningParameters) params.get(SigningEvent.IN_SIGNING_PARAMS);
                // do the signing process
                return localSigning(pluginSigningParameters);
            } else {
                log.info("Invalid event received");
                throw new PluginException(EPluginMsgs.PLUGIN_UNSUPPORTED_EVENT,
 getPluginId(), event.getEventName());
            }
        } catch (Exception e) {
            if (e instanceof PluginException) {
                throw e;
            }
            throw new PluginException(e, EPluginMsgs.PLUGIN_UNKNOWN_EXCEPTION,
 e.getMessage());
        }
    }

    @Override
    public String[] supportedEvents() {
        // only the SigningEvent is being handled
        return new String[] {
                (new SigningEvent()).getEventName()
        };
    }

    private IEventResult localSigning(PluginSigningParameters signingParams) {
        try {
```

```
            // log the 2 declared config options as an example
            log.info("value of config_option_1: " +
CONFIG_MAP.getOrDefault(config_option_1.getName(), "n/a"));
            log.info("value of config_option_2: " +
CONFIG_MAP.getOrDefault(config_option_2.getName(), "n/a"));

            // sign with SignDoc SDK (This can be completely customized)
            final Map<String, String> initMap = new HashMap<>();
            if (signingParams.getRootCertificate()!=null) {
                initMap.put(INIT_PARAM_SDSDK_ROOT_CERT,
Base64.getEncoder().encodeToString(signingParams.getRootCertificate()));
            }
            try (SignDocPDFDocument signDocPDFDocument =
SignDocPDF.createSignDocPDFDocument(ImplControlPdflib.IMPL.SDSDK,
signingParams.getDocumentContent(), initMap)) {
                Optional<SignDocPDFField> fieldOptional =
signDocPDFDocument.getFields().stream().filter(signDocPDFField ->
signDocPDFField.getName().equals(signingParams.getFieldName())).findFirst();
                if (!fieldOptional.isPresent()) {
                    throw new IllegalStateException("field " +
signingParams.getFieldName() + " not found in document");
                }

                SignDocPDFCertificateProvider certificateProvider =
new SDPPkcs12CertificateProvider(signingParams.getCertificatePkcs12(),
signingParams.getCertificatePkcs12Password());
                SignDocPDFBiometricKeyProvider biometricKeyProvider = new
SDPBiometricKeyProvider(signingParams.getBiometricKey());

                signDocPDFDocument.addSignature(certificateProvider,
signingParams.getFieldName(), signingParams.getSignatureAppearance(),
biometricKeyProvider, signingParams.getBiometricData());

                byte[] data = signDocPDFDocument.getPDFDocument();

                // return the signed PDF document
                final Map<String, Object> resMap = new HashMap<>();
                resMap.put(SigningEvent.OUT_SIGNED_DOC, data);
                resMap.put(SigningEvent.OUT_RC, SigningEvent.RETURN_CODE.SUCCESS);
                return () -> Collections.unmodifiableMap(resMap);
            }

        } catch (SDPExceptionActionNotAllowed e) {
            // thrown in case an existing signature would be overwritten
            log.diag_debug(e.getMessage(), e);
            return new SimpleEventResult(Collections.singletonMap(SigningEvent.OUT_RC,
 SigningEvent.RETURN_CODE.NOT_ALLOWED));
        } catch (SDPExceptionSignatureTooSimpleOrNotUsable e) {
            // thrown in case a signature is too simple or invalid
            log.diag_debug(e.getMessage(), e);
            return new SimpleEventResult(Collections.singletonMap(SigningEvent.OUT_RC,
SigningEvent.RETURN_CODE.SIGNATURE_TOO_SIMPLE_OR_INVALID));
        } catch (Exception e) {
            // an unexpected error
            log.error(e.getMessage(), e);
            return new SimpleEventResult(Collections.singletonMap(SigningEvent.OUT_RC,
 SigningEvent.RETURN_CODE.GENERAL_ERROR));
        }

    }

    @Override
    public String getPluginId() {
```

```
        // this is the unique plugin id
        return PLUGINID;
    }

    @Override
    public String getPluginVersion() {
        // a free to choose version string
        return "2.2.1";
    }

    @Override
    public String getDescription() {
        // a free to choose description string
        return "A sample SigningEvent handler plugin";
    }

    @Override
    public void injectPluginConfiguration(Map<String, Object> configMap) throws
 PluginException {
        // this method receives the account specific plugin configuration
        // and stores it in a local map
        this.CONFIG_MAP.clear();
        this.CONFIG_MAP.putAll(configMap);
    }

    @Override
    public String getErrorMessage(IPluginError pluginError, Locale locale) {
        // This method returns messsages for thrown PluginExceptions
        return "Error: " + pluginError.getErrorId();
    }
}
```

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 SignDocDefaultSignerHandler.

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 SignDocDefaultSignerHandler will use the assigned class to calculate the RSA signature for signature fields.

Minimal SigningRSA implementation

This is a minimal implementation of a SigningRSA plugin. This plugin uses the BouncyCastleProvider to calculate the RSA signature based on the provided certificate settings of the SignDoc account.

The complete example can be found in

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

Excerpt of an example file:

```
/**
 * This is a sample implementation of the SigningRSA plugin interface.
 * This class uses the BouncyCastleProvider to calculate the RSA signature
 * based on the provided certificate settings of the SignDoc account.
 *
 * To use the plugin it is required to...
 * 1) provide this implementation (incl. dependencies) either as class file in
 CIRRUS_HOME/plugins/classes
 *    or as jar file in CIRRUS_HOME/plugins/lib. It is not required to restart the
 SignDOc Service for this.
 * 2) make sure that the SignDocDefaultSigningHandler plugin is set as plugin
 implementation of the signing event
 *    System Settings -> Plugins -> Plugin implementations ->
 plugin.event.impl.signingevent
 * 3) To assign the full class name of this class
 (com.mycompany.plugins.MinimalRSASigner) to the setting
 *    System Settings -> Plugins -> Configuration
 *      -> SignDocDefaultSigningHandler ->
 plugin.cfg.SignDocDefaultSigningHandler.signing.rsa.class
 */
public class MinimalRSASigner extends AbstractPlugin implements SigningRSA {

    private final AtomicReference<byte[]> signingCertificate = new AtomicReference<>();

    private final List<byte[]> certificateChain = new CopyOnWriteArrayList<>();

    private final AtomicReference<RSAPrivateKey> privatekey = new AtomicReference<>();
```

```
private volatile String mError = "";

@Override
public byte[] sign(SigningSignSource aSource, int aVersion, int aHashAlgorithm) {

    String algo;
    if (aVersion == v_1_5 && aHashAlgorithm == ha_sha1)
        algo = "SHA1withRSA";
    else if (aVersion == v_1_5 && aHashAlgorithm == ha_sha256)
        algo = "SHA256withRSA";
    else if (aVersion == v_2_0) {
        mError = "OAEP not implemented";
        return null;
    } else {
        mError = "invalid argument";
        return null;
    }
    try {
        BouncyCastleProvider BCP = new BouncyCastleProvider();
        Signature signature = Signature.getInstance(algo, BCP);
        signature.initSign(privatekey.get());
        for (; ; ) {
            byte[] data = aSource.fetch(4096);
            if (data == null || data.length == 0)
                break;
            signature.update(data);
        }
        return signature.sign();
    } catch (Throwable e) {
        mError = e.getMessage();
        return null;
    }
}

@Override
public byte[] getCertificate(int aIndex) {
    return signingCertificate.get();
}

@Override
public int getCertificateCount() {
    return certificateChain.size();
}

public int getSignatureSize() {
    return (privatekey.get().getModulus().bitLength() + 7) / 8;
}

@Override
public byte[] getSigningCertificate() {
    try {
        return signingCertificate.get();
    } catch (Exception e) {
        mError = e.getMessage();
        return null;
    }
}

@Override
public String getErrorMessage() {
    return mError;
}
```

```
    @Override
    public void injectPluginConfiguration(Map<String, Object> configMap) {
        final byte[] pkcs12;
        final String pkcs12_password;
        {
            Object o = configMap.get(CONF_PKCS12_CERT);
            if (!(o instanceof byte[])) {
                throw new IllegalArgumentException(CONF_PKCS12_CERT + "=" + (o !=
null ? o.getClass().getName() : "null") + "is not of type " + byte[].class.getName());
            }
            pkcs12 = (byte[]) o;
        }
        {
            Object o = configMap.get(CONF_PKCS12_CERT_PASSWORD);
            if (!(o instanceof String)) {
                throw new IllegalArgumentException(CONF_PKCS12_CERT_PASSWORD
+ "=" + (o != null ? o.getClass().getName() : "null") + "is not of type " +
String.class.getName());
            }
            pkcs12_password = (String) o;
        }
        readPkcs12(pkcs12, pkcs12_password);
    }

    @Override
    public String getErrorMessage(IPluginError pluginError, Locale locale) {
        return "pluginError: " + pluginError.getErrorId() + " + mError: " +
getErrorMessage();
    }

    private void readPkcs12(byte[] pkcs12, String password) {
        try {
            if (password == null) {
                password = "";
            }
            BouncyCastleProvider BCP = new BouncyCastleProvider();
            KeyStore keyStore = KeyStore.getInstance("pkcs12", BCP);
            keyStore.load(new ByteArrayInputStream(pkcs12), password.toCharArray());

            if (keyStore.size() != 1) {
                throw new IllegalStateException("P12 KeyStore can have only 1 alias");
            }

            if (!keyStore.aliases().hasMoreElements()) {
                throw new IllegalStateException("P12 KeyStore has no aliases");
            }

            String certName = keyStore.aliases().nextElement();

            // (1) set private key
            RSAPrivateKey privatekey = (RSAPrivateKey) keyStore.getKey(certName,
password.toCharArray());
            this.privatekey.set(privatekey);


            final Certificate[] certificateChain =
keyStore.getCertificateChain(certName);
            if (certificateChain == null || certificateChain.length < 1) {
                throw new IllegalStateException("certificateChain is null or has no
elements");
            }

            // (2) set singing certificate
            Certificate signingCertificate = certificateChain[0];
```

```
            this.signingCertificate.set(signingCertificate.getEncoded());

            // (3) set intermediate certificates
            final List<Certificate> certificateList =
Arrays.asList(Arrays.copyOfRange(certificateChain, 1, certificateChain.length));
            this.certificateChain.clear();
            for (Certificate certificate : certificateList) {
                this.certificateChain.add(certificate.getEncoded());
            }
        } catch (IOException | KeyStoreException | NoSuchAlgorithmException |
CertificateException | UnrecoverableKeyException e) {
            mError = e.getMessage();
            throw new IllegalStateException(e.getMessage(), e);
        }
    }


    @Override
    public String getPluginVersion() {
        return "2.2.1";
    }

    @Override
    public String getDescription() {
        return "Default RSA Signer Implementation";
    }

    @Override
    public String getVendor() {
        return "Kofax";
    }

    @Override
    public String getCopyright() {
        return "(c)2009-" + Calendar.getInstance().get(Calendar.YEAR) + " Kofax
Deutschland AG";
    }

}
```

Core plugins

SignDoc default signer handler plugin

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

- Signing the PDF Digital signature field
  - By default the SignDocDefaultSignerHandler 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/v7/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
- ClamAV virus scanner
- Configure the ClamAV document scan plugin
- Test the scanning

Document scan event

To prevent invalid documents from being uploaded or viruses being spread to customers via uploaded infected documents, SignDoc Standard supports the document scan event interface.

Whenever a document or supplemental document is being uploaded, a document scan event is posted to all registered 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.
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_PROVIDER_NAME	The (localized) name of the trusted service provider implemented by this plugin.
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_CREDENTIALS_DESCRIPTION	The list of validation credentials needed by the TSP to perform a validation and/or signing. See below.

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_SIGNATURE_TYPE	The type of the digital signature requested. Currently BASIC, ADVANCED or QUALIFIED.

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 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_NAME	The name of the document to be signed.
IN_REDIRECT_URL_OK	URL to be redirected to if signing was successful.
IN_REDIRECT_URL_CANCEL	URL to be redirected to if signing has been canceled.
IN_REDIRECT_URL_ERROR	URL to be redirected to if a signing error occurred.
IN_SETTINGS	Optional. If present, the settings map will completely override the settings provided at 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.

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.

Parameter	Description
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.
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.

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_SIGNATURE_PROCESS_TOKEN	The signature token that identifies this signing process (received by PostDocumentSignatureEvent).
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.

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.

Core plugins

TSP plugin

- Digital document signing and trusted service providers
- Registering an account with Bundesdruckerei
- Configuration

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/v7/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"
    }
  ]
}
```

Chapter 5

Monitoring

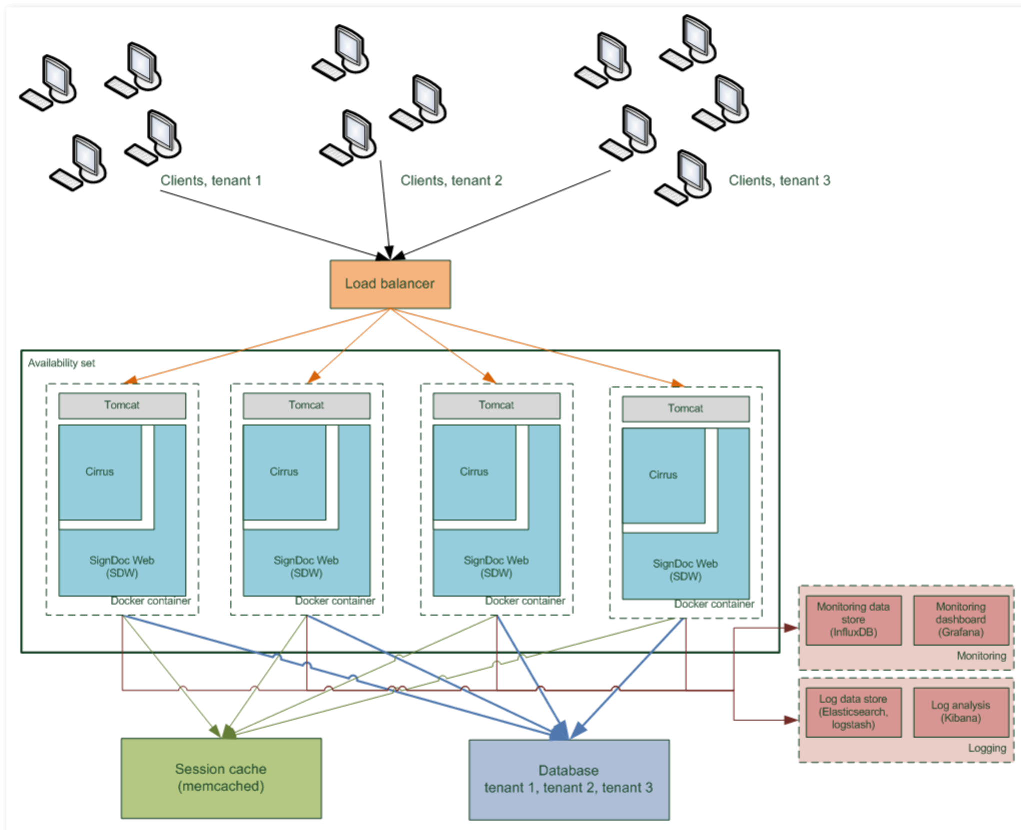
Overview

This chapter describes the current implementation of SignDoc monitoring. The metrics and protocol provided by the SignDoc implementation are shown. In addition, a sample monitoring setup using standard components (Collectd, InfluxDB, Grafana) is described.

The main design goals are:

- Providing key application metrics:
  - In addition to the usual system monitoring, key application metrics are recorded to be able to track application performance in production and detect problems in a cloud environment.
- Using well established protocols:
  - The Graphite protocol is widely used by applications to provide metrics to monitoring systems.
  - In addition to the sample monitoring server described in this document, interfaces are available to use the protocol on most monitoring systems, including DataDog, currently used by Kofax cloud operations.
  - Using standard and highly configurable components for the sample setup:
    - All components used have an established track record in application monitoring and are available at no additional cost.
  - Minimizing resource usage:
    - Metrics and naming are designed to minimize resource (notably network) usage. Metrics reporting can be filtered to exclude unwanted information.

Monitoring the KSD cluster



Monitoring protocol

Metrics are provided via the plaintext Graphite protocol.

Metrics are sent in the form:

```
<metric path> <metric value> <metric timestamp>
```

where metric path is a '.' separated list of names defining the metric.

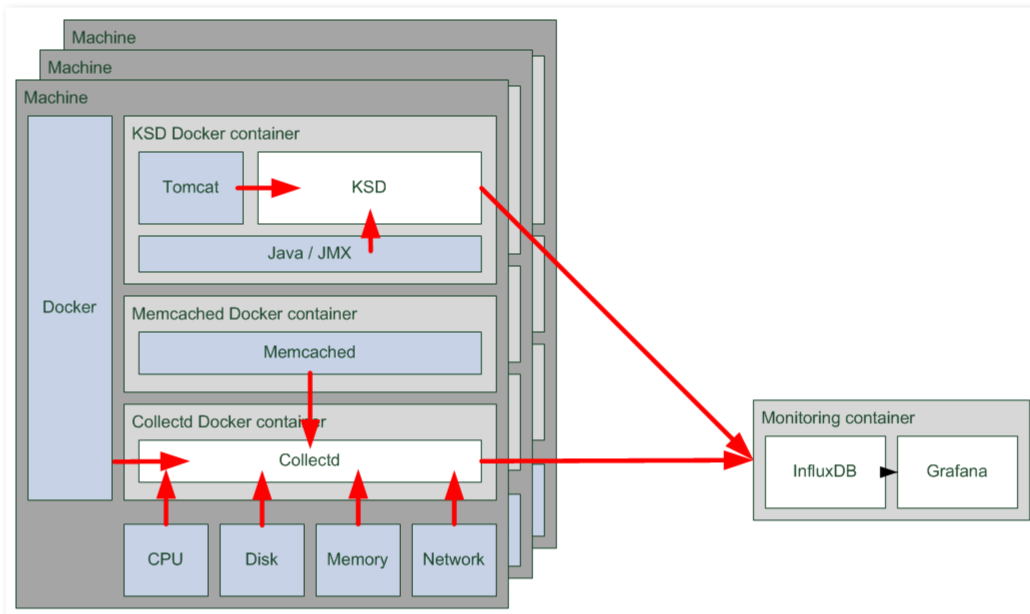
One sample metric could look like:

```
local.random.diceroll 4 1445620537
```

Data is sent by default via UDP, TCP can be configured alternatively if needed.

Each application sends the metrics at configured time intervals to the server recording the data.

Metrics being collected



Metrics can be collected on different aspects of the cluster operation:

- The computer (or VM) the application runs on
- Docker container metrics (by container)
- Java JMX metrics
- Application metrics

Application metrics and selected JMX metrics are collected by the KSD application and reported to the monitoring system.

This document also describes how system and Docker container metrics can be collected with the collectd container.

Monitoring server

Metrics can be sent to any monitoring system supporting the Graphite plaintext protocol (TCP or UDP).

This document also describes a monitoring setup using InfluxDB as a data store and Grafana for dashboard creation. This is a sample setup, not part of the product deliverable or support.

SignDoc Standard application metrics

Metric overview

Metrics are being collected for:

- Application: metrics describing the application state.
- Java JMX: selected metrics describing the Java state (memory, threads, etc.).
- Tomcat metrics via JMX: selected metrics describing the tomcat handler state.
- Docker container metrics: describing individual docker containers on a machine (cpu, memory, network, etc.).
- System metrics: describing the state of the machine the application is being run on (cpu usage, memory, disk, network i/o, etc.).

For all metrics the following description applies.

All metric names start with <instance>.<profile> where:

Item	Description
instance	The instance, or hostname, where the data is being collected.
profile	The profile of the data. • In case of system metrics, this is the category of the data (cpu, memory, disk, etc.). • In case of docker containers it is the container id or name. • In case of application metrics it is the assigned container name or id.

Metrics reporting statistical information have following endings:

Ending	Description
count	The count of events.
m1_rate	One minute exponentially weighted moving average rate of event occurrence.
m5_rate	Five minute exponentially weighted moving average rate of event occurrence.
m15_rate	Fifteen minute exponentially weighted moving average rate of event occurrence.
mean_rate	Mean rate of event occurrence.
p50	The value of the 50th percentile of the distribution.
p75	The value of the 75th percentile of the distribution.
p95	The value of the 95th percentile of the distribution.
p98	The value of the 98th percentile of the distribution.
p99	The value of the 99th percentile of the distribution.
p999	The value of the 999th percentile of the distribution.
stddev	The standard deviation of the distribution.
min	The minimum value of the distribution.
max	The maximum value of the distribution.
mean	The mean value of the distribution.

Statistical values are usually recorded in an exponentially weighted moving window. Recent events have a higher impact on the statistical values.

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.

Starting with the release 1.3, some configuration options have been moved from configuration files to the configuration service.

Monitoring and the metrics processed can be configured in cirrus.properties configuration file with the following properties:

- **monitoring.host** The hostname or IP address of the monitoring server the metrics are sent to. If left empty, no monitoring data is sent (default).
- **monitoring.port** The port number where monitoring data is sent to. Defaults to 2003
- **monitoring.protocol** The protocol being used (TCP or UDP). Defaults to UDP.
- **monitoring.filter.include** Regular expression to specify which metrics form the available list should be reported. Defaults to all.
- **monitoring.filter.exclude** Regular expression specifying which metrics should be excluded form monitoring. By default, following statistic information is being excluded: count, m5_rate, m15_rate, max, min, mean_rate, p50, p75, p95, p98, p99, p999, stddev

Metric description

Requests

HTTP request data for the two types of requests processed: cirrus and rest.

<instance>.<profile>.cirrus.req.<req_type>.count

<instance>.<profile>.cirrus.req.<req_type>.m1_rate

<instance>.<profile>.cirrus.req.<req_type>.m5_rate

<instance>.<profile>.cirrus.req.<req_type>.m15_rate

<instance>.<profile>.cirrus.req.<req_type>.mean_rate

<instance>.<profile>.cirrus.req.<req_type>.p50

<instance>.<profile>.cirrus.req.<req_type>.p75

<instance>.<profile>.cirrus.req.<req_type>.p95

<instance>.<profile>.cirrus.req.<req_type>.p98

<instance>.<profile>.cirrus.req.<req_type>.p99

<instance>.<profile>.cirrus.req.<req_type>.p999

<instance>.<profile>.cirrus.req.<req_type>.stddev

<instance>.<profile>.cirrus.req.<req_type>.mean

<instance>.<profile>.cirrus.req.<req_type>.min

<instance>.<profile>.cirrus.req.<req_type>.max

Response status

HTTP response statistics by status code.

<instance>.<profile>.cirrus.resp.<resp_type>.<resp_status>.count

<instance>.<profile>.cirrus.resp.<resp_type>.<resp_status>.m1_rate

<instance>.<profile>.cirrus.resp.<resp_type>.<resp_status>.m5_rate

<instance>.<profile>.cirrus.resp.<resp_type>.<resp_status>.m15_rate

<instance>.<profile>.cirrus.resp.<resp_type>.<resp_status>.mean_rate

Where:

Item	Description
resp_type	The response type. Currently rest and cirrus.
resp_status	The group of status codes: httpStatus2XX: 200 – 299 status codes httpStatus3XX: 300 – 399 status codes httpStatus4XX: 400 – 499 status codes httpStatus5XX: 500 – 599 status codes httpStatusXXX: all other status codes

REST API calls

Metrics describing the number and duration of REST API calls.

<instance>.<profile>.rest.<ver>.<controller>.<function>.count

<instance>.<profile>.rest.<ver>.<controller>.<function>.m1_rate

<instance>.<profile>.rest.<ver>.<controller>.<function>.m5_rate

<instance>.<profile>.rest.<ver>.<controller>.<function>.m15_rate

<instance>.<profile>.rest.<ver>.<controller>.<function>.mean_rate

<instance>.<profile>.rest.<ver>.<controller>.<function>.p50

<instance>.<profile>.rest.<ver>.<controller>.<function>.p75

<instance>.<profile>.rest.<ver>.<controller>.<function>.p95

<instance>.<profile>.rest.<ver>.<controller>.<function>.p98

<instance>.<profile>.rest.<ver>.<controller>.<function>.p99

<instance>.<profile>.rest.<ver>.<controller>.<function>.p999

<instance>.<profile>.rest.<ver>.<controller>.<function>.stddev

<instance>.<profile>.rest.<ver>.<controller>.<function>.mean

<instance>.<profile>.rest.<ver>.<controller>.<function>.min

<instance>.<profile>.rest.<ver>.<controller>.<function>.max

For each API call following fields are set:

Field	Description
ver	The REST API version.
controller	The REST API controller: a: account controller d: document controller p: signing package controller r: reminder controller s: signer controller (currently unused) t: team controller u: user controller y: system controller
function	The function being called.

Queues

Selected data on the internal ActiveMQ message queues used by the application: task and notification.

- Average amount of time, in milliseconds, that messages sat in the queue before being consumed: `<instance>.<profile>.cirrus.queue.<queue_type>.averageEnqueueTime`
- Number of messages that have been acknowledged and removed from the queue: `<instance>.<profile>.cirrus.queue.<queue_type>.dequeueCount`
- Percentage of available memory in use: `<instance>.<profile>.cirrus.queue.<queue_type>.memoryPercentUsage`

JMX metrics

This section describes the selection of Java JMX metrics made available by the application. For the description refer to the Java JMX documentation.

Memory

<instance>.<profile>.jmx.memory.heap.committed

<instance>.<profile>.jmx.memory.heap.init

<instance>.<profile>.jmx.memory.heap.max

<instance>.<profile>.jmx.memory.heap.usage

<instance>.<profile>.jmx.memory.heap.used

<instance>.<profile>.jmx.memory.non-heap.committed

<instance>.<profile>.jmx.memory.non-heap.init

<instance>.<profile>.jmx.memory.non-heap.max

<instance>.<profile>.jmx.memory.non-heap.usage

<instance>.<profile>.jmx.memory.non-heap.used

<instance>.<profile>.jmx.memory.total.committed

<instance>.<profile>.jmx.memory.total.init

<instance>.<profile>.jmx.memory.total.max

<instance>.<profile>.jmx.memory.total.usage

<instance>.<profile>.jmx.memory.total.used

Threads

<instance>.<profile>.jmx.thread.daemonThreadCount

<instance>.<profile>.jmx.thread.peakThreadCount

<instance>.<profile>.jmx.thread.threadCount

<instance>.<profile>.jmx.thread.totalStartedThreadCount

Garbage collection

<instance>.<profile>.jmx.gc.MarkSweep.collectionCount

<instance>.<profile>.jmx.gc.MarkSweep.collectionTime

<instance>.<profile>.jmx.gc.Scavenge.collectionCount

<instance>.<profile>.jmx.gc.Scavenge.collectionTime

Tomcat processor metrics

A selection of the Tomcat handler (global processor) metrics are made available, for each of the handlers configured.

<instance>.<profile>.catalina.processor.<handler_name>.bytesReceived

<instance>.<profile>.catalina.processor.<handler_name>.bytesSent

<instance>.<profile>.catalina.processor.<handler_name>.errorCount

<instance>.<profile>.catalina.processor.<handler_name>.maxTime

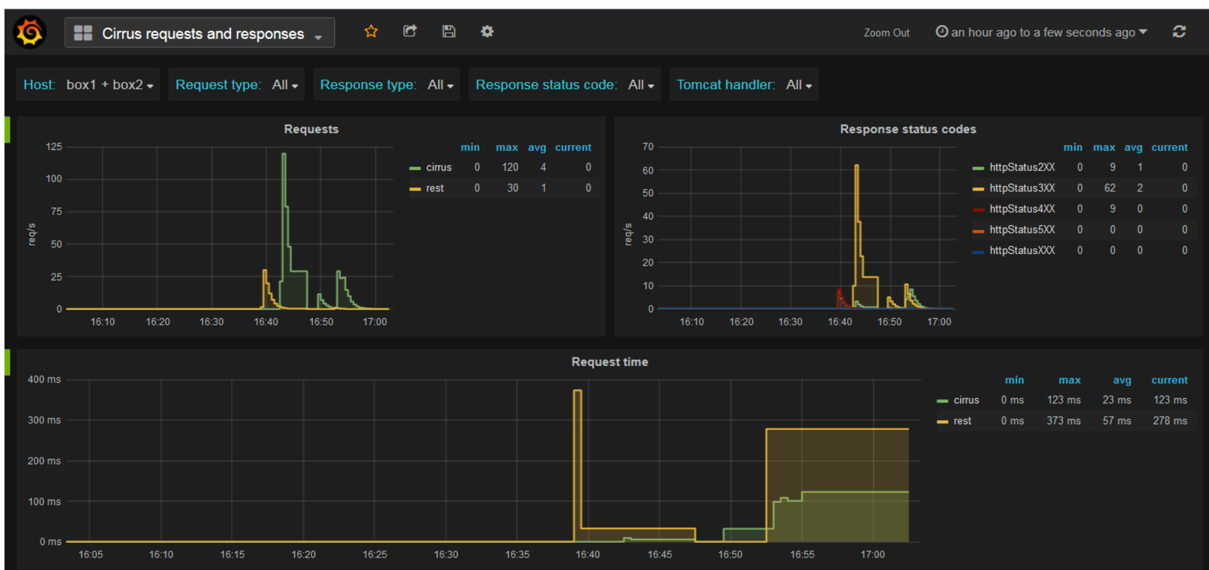
<instance>.<profile>.catalina.processor.<handler_name>.processingTime

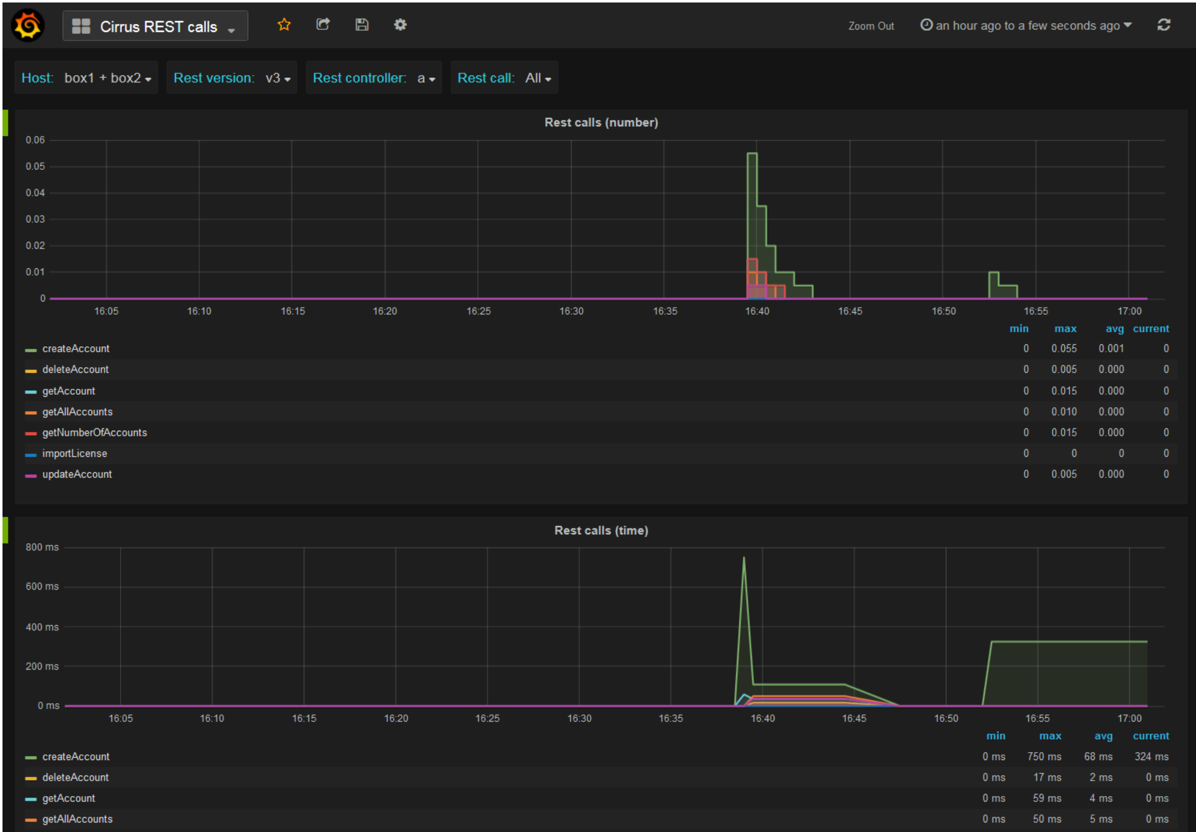
<instance>.<profile>.catalina.processor.<handler_name>.requestCount

Example setups

The setups described in this chapter are provided as an example, without being part of the product or support. They can be found on the Kofax SignDoc Standard product downloads page.

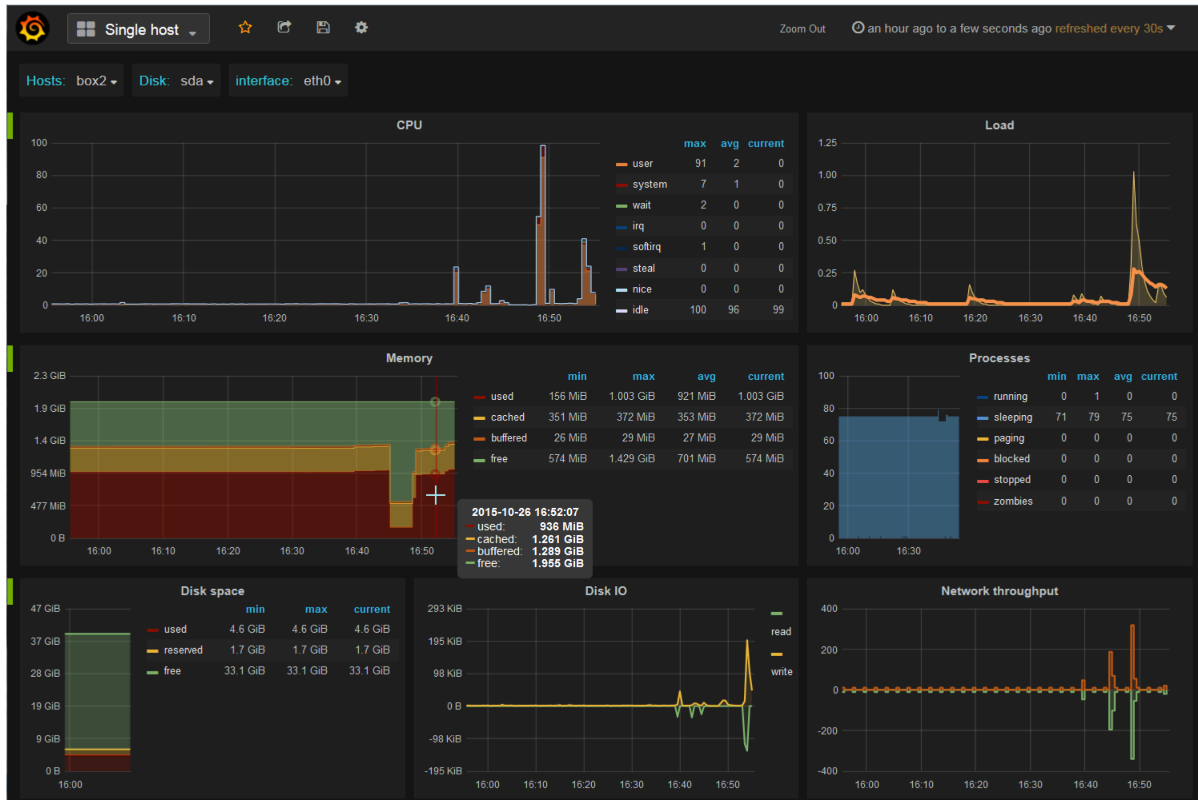
Monitoring service sample setup

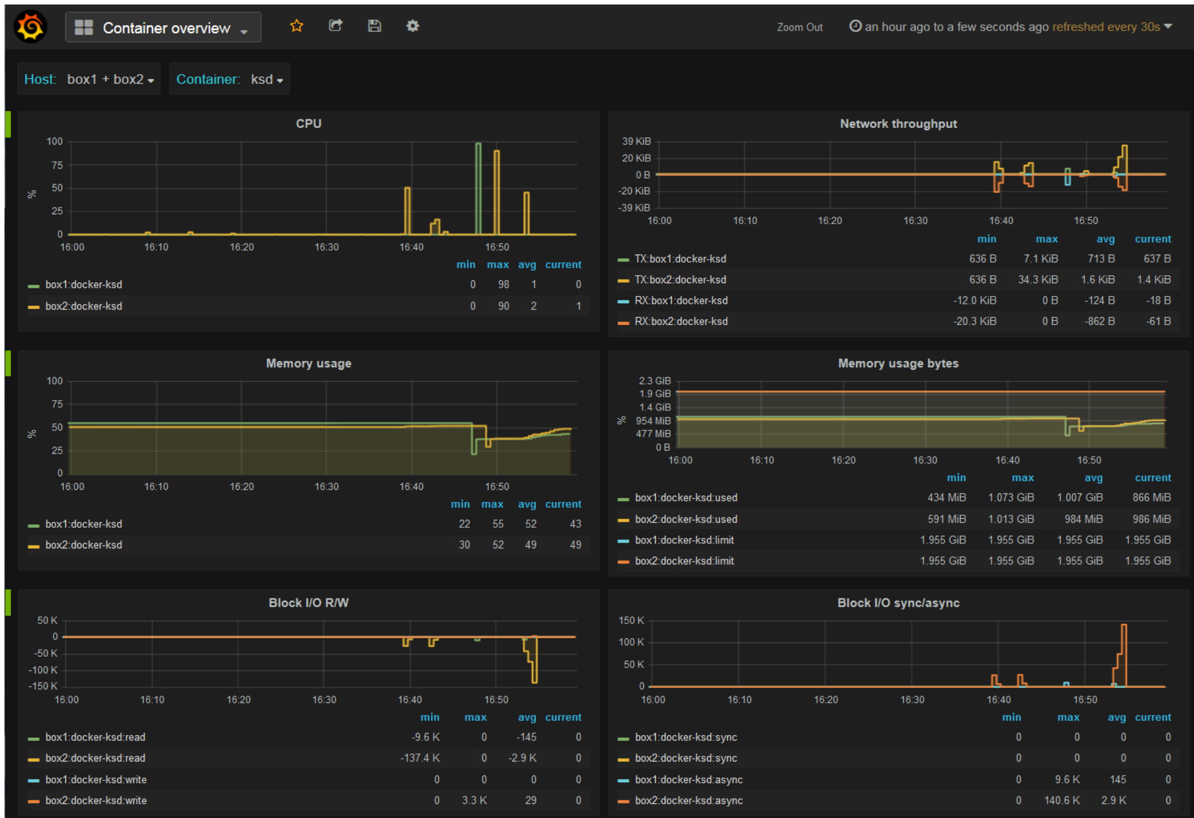




Collectd sample setup for collecting system metrics

This is an example how Collectd (via Docker containers) can be used to collect system metrics.


Single host
Zoom Out
an hour ago to a few seconds ago refreshed every 30s
Hosts: box2
Disk: sda
interface: eth0
CPU
max avg current
user 91 2 0
system 7 1 0
wait 2 0 0
irq 0 0 0
softirq 1 0 0
steal 0 0 0
nice 0 0 0
idle 100 96 99
Load
Memory
min max avg current
used 156 MiB 1.003 GiB 921 MiB 1.003 GiB
cached 351 MiB 372 MiB 353 MiB 372 MiB
buffered 26 MiB 29 MiB 27 MiB 29 MiB
free 574 MiB 1.429 GiB 701 MiB 574 MiB
2015-10-26 16:52:07
used: 936 MiB
cached: 1.261 GiB
buffered: 1.289 GiB
free: 1.955 GiB
Processes
min max avg current
running 0 1 0 0
sleeping 71 79 75 75
paging 0 0 0 0
blocked 0 0 0 0
stopped 0 0 0 0
zombies 0 0 0 0
Disk space
min max current
used 4.6 GiB 4.6 GiB 4.6 GiB
reserved 1.7 GiB 1.7 GiB 1.7 GiB
free 33.1 GiB 33.1 GiB 33.1 GiB
Disk IO
read
write
Network throughput



Glossary

Cirrus

The internal name of the SignDoc Standard part of the project. Manages signing packages, workflow and account / user administration.

cluster

The sum of instances working together as a whole to provide the application functionality. Consists of the application service instances, but also load balancer, cache service, log and monitoring service, etc.

Collectd

System statistics collection daemon https://collectd.org

container

Virtualization unit that encapsulates all necessary components to run an application on a system. In this document the term refers to Docker containers.

Grafana

Dashboard builder for visualizing monitoring data http://grafana.com

Graphite

Set of monitoring components used to process monitoring information. In this document used to refer to the protocol defined by said components to transmit metrics http://graphite.readthedocs.org/en/latest/index.html

InfluxDB

Distributed time-series database used to store monitoring data https://influxdb.com

metric

A value to be monitored that is composed of name, value and timestamp.

on premise deployment

Form of deployment where the customer maintains control of all hardware and software components used to deploy the application. Only the application is provided, the customer administers the database server, servlet container, etc.

private cloud deployment

Form of cloud deployment where all resources of a cluster are allocated exclusively to one customer (tenant).

public cloud deployment

Form of cloud deployment where multiple customers (tenants) share the resources of one cluster.

tenant

A customer or an organization that uses the services of an application and has multiple users. Multiple tenants can be set up on one system. The users of one tenant cannot access data of another tenant.

Chapter 6

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 7

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 "
```