



Kofax SignDoc Web Developer's Guide

Version: 3.2.0

Date: 2022-08-03

© 2022 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.....	6
Related documentation.....	6
Training.....	7
Getting help with Kofax products.....	7
SignDoc Web features.....	8
Chapter 1: Functional structure of SignDoc Web.....	11
Architecture.....	11
Important files and directories.....	12
Communication with SignDoc Web server.....	16
Chapter 2: Integration.....	17
Preload and prepare web service.....	17
Example UploadAndPrepareDocument.....	19
Integration in existing web applications.....	20
Start SignDoc Web from external web application.....	20
Request types.....	23
Request parameters.....	23
Create or update a field in the document.....	25
Add text to a document.....	25
Insert a form field by coordinates.....	27
Insert a form field with position located by text phrase.....	28
Insert a general capture field.....	29
Update the value and attribute of an existing form field or signature field.....	30
Convert an existing form field to a signature field.....	31
Set metadata.....	32
Signature archive interaction.....	33
Field validation before update.....	33
Field change during validation.....	34
Signature watermark.....	36
Remove an existing form field or signature field.....	40
SignDoc Web field locking.....	40
URI syntax.....	42
Document metadata.....	44
Signature and form fields.....	44
Radio buttons.....	45

Example of document load page.....	47
Dynamic tablet screens.....	50
Chapter 3: Remote interface.....	54
Same-origin policy.....	54
Execute actions in mobile-gui.....	55
Notification about action events.....	63
mobile-app document sequence example.....	65
Hints and examples.....	66
Version history.....	70
Dictionary.....	73
Chapter 4: REST interface.....	74
REST URL.....	77
REST error response.....	77
REST API reference v5.....	78
Preload PDF document with commands and prepare options.....	79
Activate preloaded PDF document for processing.....	81
Append PDF document to previously loaded document.....	83
Attach PDF document to previously loaded document.....	84
Remove attachment from previously loaded document.....	85
Get document information.....	86
Get document page image.....	92
Get document.....	94
Get audit logs of document.....	95
Add signature.....	96
Insert signature field.....	102
Update signature field.....	105
Clear signature field.....	109
Insert text field.....	109
Update text field.....	113
Insert checkbox field.....	116
Update checkbox field.....	119
Delete document field.....	123
Archive document.....	124
Remove document.....	125
Document coordinate system.....	126
Chapter 5: Plugin interface.....	127
SignDoc Web plugins general information.....	127
Available plugin interfaces.....	131

General SignDoc Web plugin interface.....133

Trusted service provider..... 136

Chapter 6: Script plugins..... 141

Chapter 7: Online signature verification enhancements..... 143

Preface

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

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

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

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

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

Related documentation

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

https://docshield.kofax.com/Portal/Products/en_US/SD/3.2.0-f97v3tx5n6/SD.htm

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

Release notes

- *Kofax SignDoc Release Notes*

Technical specifications

- *Kofax SignDoc Technical Specifications*

Guides

- *Kofax SignDoc Web Administrator's Guide*

Help

- *Kofax SignDoc Web Help*
- *Kofax SignDoc Device Connector Help*

Software development kit

- *Kofax SignDoc SDK API Documentation (C)*
- *Kofax SignDoc SDK API Documentation (C++)*
- *Kofax SignDoc SDK API Documentation (.NET with exceptions)*
- *Kofax SignDoc SDK API Documentation (.NET without exceptions)*
- *Kofax SignDoc SDK API Documentation (Java)*

Training


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

Getting help with Kofax products

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

To access the Kofax Knowledge Base:

1. Go to the [Kofax website](#) home page and select **Customers**.
2. When the Customers page appears, select **Knowledge Base**.

 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.

From the Knowledge Base home page, you can:

- Access the Kofax Community (for all customers).
Click the **Community** link at the top of the page.
- Access the Kofax Customer Portal (for eligible customers).
Click the **Support** link at the top of the page. When the Customer & Partner Portals Overview appears, click **Log in to the Customer Portal**.
- Access the Kofax Partner Portal (for eligible partners).
Click the **Support** link at the top of the page. When the Customer & Partner Portals Overview appears, click **Log in to the Partner Portal**.
- Access Kofax support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.
Click **Support Details**, and then select the appropriate tab.

SignDoc Web features

- **Browser-independent**
Simple web-based user interface. Support for most common browsers on PCs (Mozilla Firefox, Google Chrome, Microsoft Edge).
- **Content protection**
Protect the integrity of documents by sealing them with a digital signature. Supports any x509v3 certificate provided through the key store (e.g. Windows certificate store) or through plugin interface to digitally sign documents by using an external method (e.g. for usage of a Hardware Security Module - HSM).
- **Customer-specific data**
Set customer metadata for workflow process.
- **Customer-specific document processing**
Use the plugin interface to write your own plugins for document processing.
- **Customer-specific labels**
Customize labels of signature fields depending on the document workflow.
- **Customization**
Enterprises can utilize SignDoc Web and launch their own e-signing solution, as well as integrate the functionality of SignDoc Web into their own apps (SDK available). Using your enterprise .css style is also supported.
Additionally, for user attendance to most important document content signature device (e.g. sign pad devices) background can be dynamically updated with document specific information.
- **Device-independent**
Supports many different signature pads, interactive pen displays, tablet/slate PCs, iPad, Android and Windows tablets and iOS, Android and Windows smartphones.
- **Document binding**

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

- **Enter signature fields**

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

- **Fill out and sign PDF forms**

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

- **Flatten PDF**

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

- **Guidance in the signing process**

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

- **Handwritten signature capturing**

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

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

- **Identity/signature verification**

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

- **Integration**

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

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

- **Offline**

Take documents offline.

- **On-premises or cloud-based**

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

- **PKI-based certificates**

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

- **Print**

Print a document before or after saving or signing.

- **Signature capturing**

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

- **Use PDF templates**

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

- **Watermark**

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

Chapter 1

Functional structure of SignDoc Web

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

Architecture

Desktop clients

A variety of browsers are supported as desktop clients on Windows via the [Kofax SignDoc Device Support](#).

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

Mobile clients

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

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

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

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

SignDoc Web server

- Layer 1 - Operating systems
SignDoc Web can be installed on Windows and Linux operating systems with a 32 or 64bit architecture
- Layer 2 - SignDoc Web application
The application consists of a WAR (Web Application Archive) which is deployed into a Web Application Server. The WAR file contains Native Libraries which are used for licensing and PDF handling. All the configuration files are stored outside of the WAR file in a folder called SDWEB_HOME. SignDoc Web also offers a plugin interface for possible means of extension.

- Layer 3 - SOAP & REST interfaces

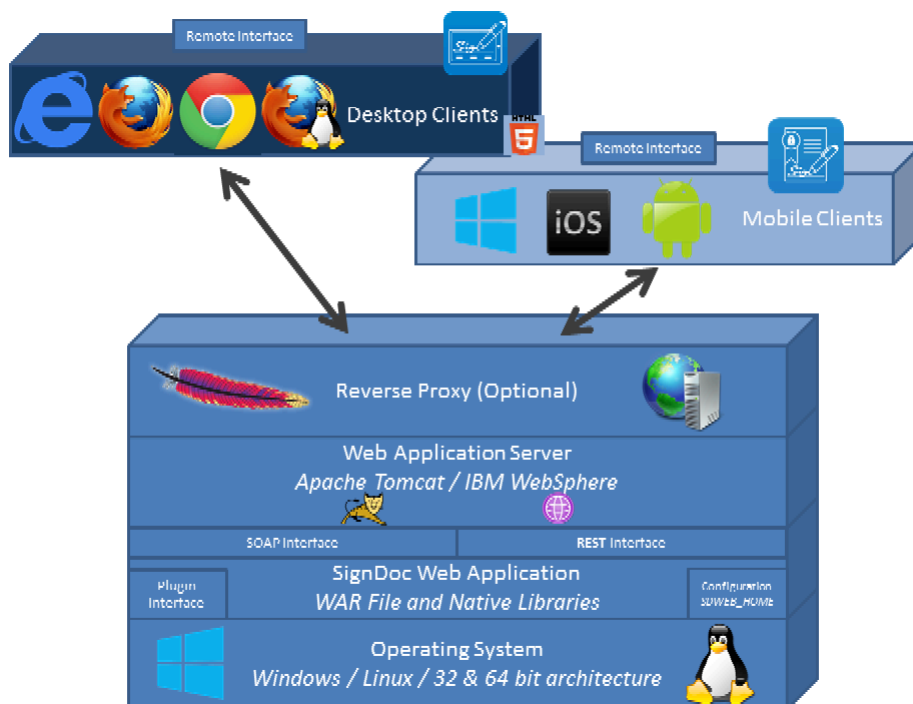
Documents can be uploaded via a web service and then accessed by using a Reference URL. It is also possible to upload and perform operations on documents via a REST interface.

- Layer 4 - Web application server

SignDoc Web runs on the standard Web Application Server Apache Tomcat, for which *Kofax SignDoc Installation Guide* offers support. Installation on other JEE compliant servers is not supported. For prerequisites, see *Kofax SignDoc Technical Specifications* document.

- Layer 5 - Reverse proxy (optional)

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



Important files and directories

SignDoc Web consists of two parts:

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

Directory `SDWEB_HOME`

In this section you can find an overview of the files and directories which are provided as part of the `SDWEB_HOME` directory.

- `c2s`
Click-to-Sign templates
- `conf`

Configuration files

- `dms`
File-based DMS directory (created upon first use)
- `dms/de.softpro.sdweb.plugins.impl.FileDms`
SOFTPRO file DMS directory
- `doctemplates`
Document templates
- `fonts`
Font configuration
- `i18n`
Global translation files (Internationalization)
- `interfaces`
Plugin interfaces, examples and documentation
- `logs`
Log files (Audit, Error, Performance)
- `plugins`
Non-Kofax plugins
- `preloaded_docs`
Preloaded documents
- `resources`
Resource directory for public key to encrypt biometric data of signatures and other configurable elements like icons.
- `tablet_screens`
Tablet Screens (Backgrounds)
- `tools`
TabletScreenImageCreator and PasswordEncryptionHelper

Directory conf

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

i If there is a need to run multiple SignDoc Web servers on one machine and configure them separately, see the description of the instance-based configuration in the "Multi-instance configuration" section of the *Kofax SignDoc Web Administrator's Guide*.


- `cert_store.p12`
A custom PKCS#12 file containing a certificate that is used for the digital signing of the PDF document.
- `cert_store_readme.txt`
Readme file explaining the usage of certificates.
- `configuration.xml`
This is a bootstrap configuration file of the rich client which shouldn't be changed by a customer.
- `language.properties`

This is the default language properties file used by the rich client if no language file is available for the locale.

- `language_xx.properties`
These files contain all labels and messages which are visible in the rich client. The actual language file is chosen by the Java locale mechanism.
- `mobile_configuration.xml`
In this rich client configuration file the functionality and features of the mobile GUI can be configured.
- `sample_ssl_cert.pfx`
SSL certificate file.
- `sdcustom.properties`
In this rich client configuration file the client logging and the supported locales can be configured.
- `sdweb_config.groovy`
This file is the main configuration file for the server. It is written in a compact and structured syntax. It can be used in an instance-based configuration scenario to have a different server configuration for each instance.
- `server_configuration.xml`
This is a bootstrap configuration file of the rich client which shouldn't be changed by a customer.
- `signdoc_configuration.xml`
In this rich client configuration file the functionality and features of the desktop GUI can be configured.
- `signdoc-logger.properties`
A standard configuration file that can be changed for customized logging output.
- `tomcat-logging.properties`
A properties file that contains the Tomcat logging configuration.

Directory doctemplates

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

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

- Sample PDF documents with generic capture and/or image capture fields.
`lorem_ipsum_4_pages.pdf`
`TrapezaOpenJointAccounts.pdf`

Directory interfaces

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

- `NBC2SSignatureRendererSample`
An example of an alternative Click-to-Sign image renderer.
- `PreparePlugin`

A simple SignDoc Web Prepare plugin. The plugin automatically inserts a signature field in the bottom left corner of the first page. Additionally to this, it will populate each text field with the current date and check each checkbox form element.

- `ReadOnlyDmsPlugin`
Simple DMS plugin demonstrating, how to make interactive fields readonly before storing the document in the DMS.
- `SetDocIdPlugin`
This DMS plugin demonstrates, how a DMS plugin can set the document ID.
- `SignatureArchivePlugin`
A simple Signature Archive plugin, that stores all captured biometric signatures in the file system using the user ID as index.
- `SignatureVerificationPlugin`
A plugin demonstrating the possibility to capture a signature reference and afterward's validating against it.
- `SignPKCS7Demo`
A plugin demonstrating the possibility to sign documents using an external method.
- `VSVTestPlugin`
Combined Prepare and Signature Verification plugin, that will display a dialog with the captured signature and the signature of Albert Einstein as reference signature.
- `ZDms`
A simple DMS plugin saving documents to a directory.
- `ZPrepare`
A prepare plugin which inserts signature fields based on location, textphrase and parts of the document ID.

Directory logs

The `logs` directory contains the SignDoc Web log files and subdirectories.

Directory tablet_screens

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

- Sample Tablet Screens available with SignDoc Web.
 - `default.xml`
 - `dynamic_content_example.xml`
 - `next_screen.xml`
 - `ok_screen.xml`
 - `piggybank_example.xml`
 - `WacomSTUSeries.xml`
- Test pages to test Dynamic Content displayed on the SignPad.
 - `dynamic_content_example.html`
 - `piggybank_example.html`
 - `Piggybank_200.png`
- XML Schema of Tablet Screens

TabletScreenLayout.xsd

Communication with SignDoc Web server

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

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

Chapter 2

Integration


Preload and prepare web service

Description

The preload web service (SOAP) allows a user of SignDoc Web to upload and prepare a document before the document is actually opened in the GUI. It is possible to specify preparation commands for the document when uploading, so that the document is completely prepared for immediate usage. This removes the need to prepare the document via the load/doc servlet. The web service returns a reference ID `ref_id`, action URL `action_url` and load URL `redir_url` back to the caller. The latter URL can be used in any link to open the document via a simple GET request.

The default implementation of the SignDoc Web PreloadPlugin is:

`de.softpro.sdweb.plugins.impl.SimpleFilePreloader`

 After the document was displayed in SignDoc Web using the `ref_id` the `ref_id` is invalid. This means that `ref_id` can only be used once to load a document.

Useful tools

Free tools:

- SOAP Test Tool
 - soapUI www.soapui.org/ (take the free version)
- Java
 - Eclipse www.eclipse.org/
 - Netbeans netbeans.org/
- C/C++/.Net
 - Visual Studio msdn.microsoft.com/vstudio (take the Express version)

WSDL

The WSDL can be accessed via the URL (also attached as `preload.wsdl`)

`http(s)://<server>:<port>/sdweb/services/preload?wsdl`

Example

```
localhost:6610/sdweb/services/preload?wsdl
```

The web service offers multiple operations of which only the operation UploadAndPrepareDocument is relevant, since it is a superset of the other operations. Only the relevant operation (UploadAndPrepareDocument) is described in this document. The other operations (UploadDocument and UploadDocumentEx) are obsolete.

UploadAndPrepareDocument

- Request: UploadAndPrepareDocumentRequest
- Response: UploadAndPrepareDocumentResponse

UploadAndPrepareDocumentRequest

This request allows to upload and prepare a document in the server. The actual document handling on server side is done with an IPreload plugin.


Parameters

Name	XML Type	Required	Description
docdata64	xs:string	yes	Contains the binary document in BASE64 encoded format
cmd	sequence of xs:string	no	Contains commands to prepare the document. Examples Insert missing Signature fields, prefill form data, set metadata
option	sequence of tns:keyValuePair	no	Contains additional document attributes that should be set by the server. The attributes have the same semantic as the corresponding servlet attributes. Examples dmsid (DMS plugin), docid (Document ID), resulturl (Result URL)

UploadAndPrepareDocumentResponse

This is the response to the UploadAndPrepareDocumentRequest. In Error case a wsdl:fault message of type tns:ServiceException is thrown. The error case is available in the fault message. In success case a preloadResult is returned.

preloadResult contents

Name	XML Type	Description
ref_id	xs:string	Contains the unique ID of the preloaded document. <div> This is not the document ID.</div>
action_url	xs:string	The URL to the default load servlet of the SignDoc Web Server.
redir_url	xs:string	The complete URL to open the document via simple GET request. By using this URL, the preloaded document will be opened and displayed by the SignDoc Web Server.

Examples

SOAP request

See [UploadAndPrepareDocument](#) for a full example request.

Example request description:

- docdata64 is a simple PDF document (1 page) without any signature fields
- A signature field is inserted in the lower left corner of the PDF document
- The dimension and position is defined using document coordinates
- The name of the field is SignatureFixed
- multiple signature fields are inserted where the position is determined by the location of a text phrase
- The search text is: "rhoncus." (without the quotes)
- The dimension is defined using document coordinates
- The exact position is defined by offsets to the individual found positions.
- The name of the inserted field starts with "SignatureAuto" and is suffixed by a number (SignatureAuto_1, SignatureAuto_2 ,...)
- The document ID (docid) is set to 12345
- The DMS plugin (dmsid) to be used with this document is set to:
de.softpro.sdweb.plugins.impl.FileDms

SOAP response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:UploadAndPrepareDocumentResponse xmlns:ns2="http://sdweb/">
      <preload_result>
        <ref_id>1333011980416_5adc6e02-1d70-48fc-8fdf-e6333b0157d7</ref_id>
        <action_url>http://localhost:6610/sdweb/load/doc</action_url>
        <redir_url>http://localhost:6610/sdweb/load/doc?
ref_id=1333011980416_5adc6e02-1d70-48fc-8fdf-e6333b0157d7</redir_url>
      </preload_result>
    </ns2:UploadAndPrepareDocumentResponse>
  </soap:Body>
</soap:Envelope>
```

Example UploadAndPrepareDocument

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:sdw="http://sdweb/" xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/">
<soapenv:Header/>
  <soapenv:Body><sdw:UploadAndPrepareDocument>
    <!--The PDF document in base64 encoding-->
    <docdata64>
      JVBERi0xL
      ...
      KJSVFT0YK
    </docdata64>
    ...
    <!-- commands to prepare the document (optional) -->
    <cmd>name=SignatureFixed|page=1|bottom=10|left=10|width=100|height=50|
type=formfield|subtype=signature</cmd>
    <cmd>name=SignatureAuto|searchtext=rhoncus.|width=140|height=50|offsetx=-45|
offsety=30|type=formfield|subtype=signature</cmd>
    <cmd>name=Test|value=testmetadata|type=metadata</cmd>
```

```
<!-- document options (optional) -->
<option>
  <key>docid</key>
  <value>12345</value>
</option>
<option>
  <key>dmsid</key>
  <value>de.softpro.sdweb.plugins.impl.FileDms</value>
</option>
</sdw:UploadAndPrepareDocument>
</soapenv:Body>
</soapenv:Envelope>
```

Integration in existing web applications

Start SignDoc Web from external web application

SignDoc Web can open PDF documents using different methods and sources.

Servlets for opening or loading a document

- Upload a document to the server
- The server downloads a document by URL
- Use a predefined template
- Use DMS

Upload a document to the server

Servlet name:	load/byupload
Supported request types:	multipart/form-data
Load parameter:	docdata Contains the binary data of the document. This parameter must be part of a multipart/form-data servlet request. cmd Optional. [0..n] parameters of this type. For more details see Create or update a field in a document .

Example

```
<form action="http://localhost:6610/sdweb/load/byupload" target="_blank"
  enctype="multipart/form-data" method="post">
  <input type="file" name="docdata" maxlength="255" size="50"/><br/>
  <input type="submit" name="opendoc" value="open document"/>
  <!-- optional parameters -->
  <!-- defines a unique document ID -->
  <input type="hidden" name="docid" value="a_doc_id" id="docid" />
</form>
```

The server downloads a document by URL

Servlet name:	load/byurl
Supported request types:	HTTP POST/GET, URL
Load parameter:	<p>docurl</p> <p>A valid URL pointing to a document to download.</p> <p>The URL should be in URL Encoded format URLEncoded, URL Encode Page. SignDoc Web supports HTTP and HTTPS downloads.</p> <p>cmd</p> <p>Optional. [0..n] parameters of this type.</p> <p>For more details see Create or update a field in a document.</p>

Example

The server downloads a document.

Manual form request - opens in new window:

```
<form action="http://localhost:6610/sdweb/load/byurl" target="_blank" method="post">
  <input type="text" name="docurl" maxlength="255" size="50" value=""/><br/>
  <input type="submit" name="opendoc" value="open document"/>
  <!-- optional parameters -->
  <!-- defines a unique document ID -->
  <input type="hidden" name="docid" value="a_doc_id" id="docid" />
</form>
```

Prefilled request - opens the PDF Reference Specification (caution 10 MByte, >1200 pages) in new window:

```
<form action="http://localhost:6610/sdweb/load/byurl" target="_blank" method="post">
  <input type="hidden" name="docurl" maxlength="255" size="50" value="http://
partners.adobe.com/public/developer/en/pdf/PDFReference16.pdf" />
  <input type="submit" name="opendoc" value="open document"/>
  <!-- optional parameters -->
  <!-- defines a unique document ID -->
  <input type="hidden" name="docid" value="a_doc_id" id="docid" />
</form>
```

The same as link:

```
<a href="http://localhost:6610/sdweb/load/byurl?docurl=http%3A%2F%2Fpartners.adobe.com%2Fpublic%2Fdeveloper%2Fen%2Fpdf%2FPDFReference16.pdf">open document</a>
```

Use a predefined template

Servlet name:	load/bytemplate
Supported request types:	HTTP POST/GET, URL

Load parameter:	template A template available in the SignDoc Web server. Templates are PDF documents, usually having form elements in the document, that can be automatically or interactively prefilled/edited. By default document templates are located in the directory: %SDWEB_HOME%/doctemplates cmd Optional. [0..n] parameters of this type. For more details see Create or update a field in a document .
-----------------	---

Example

Manual template selection from a dropdown box - opens in new window:

```
<form action="http://localhost:6610/sdweb/load/bytemplate" target="_blank"
method="post" name="templatechooser" id="templatechooser" >
  <select name="template">
    <option value="">Please choose...</option>
    <option value="Invoice.pdf" >Invoice.pdf</option>
  </select>
  <input type="submit" name="opendoc" value="open document"/>
  <!-- optional parameters -->
  <!-- defines a unique document ID -->
  <input type="hidden" name="docid" value="a_doc_id" id="docid" />
</form>
```

Prefilled request - opens the template Invoice.pdf in new window:

```
<form action="http://localhost:6610/sdweb/load/bytemplate" target="_blank"
method="post">
  <input type="hidden" name="template" value="Invoice.pdf" />
  <input type="submit" name="opendoc" value="open document"/>
  <!-- optional parameters -->
  <!-- defines a unique document ID -->
  <input type="hidden" name="docid" value="a_doc_id" id="docid" />
</form>
```

The same as link:

```
<a href="http://localhost:6610/sdweb/load/bytemplate?
template=Invoice.pdf">open document</a>
```

Use DMS

Servlet name:	load/bydms
Supported request types:	HTTP POST/GET, URL
Load parameter:	docid A unique ID for the document to be loaded. cmd Other optional parameters like the cmd statements can be used, [0..n] parameters of this type. For more details see Create or update a field in a document .

Example

`http://localhost:6610/sdweb/load/bydms?docid=01234356789`

Request types

SignDoc Web supports different request types to load a document:

- HTTP form request
- URL request

HTTP form request

General syntax:

```
<form action="http[s]://<server_name>[:server_port]/sdweb/<load_servlet>"
  method="[post|get]">
  <!-- add additional parameters to the request -->
  <input type="hidden" name="<parameter_name>" value="<parameter_value>" />
</form>
```

URL request

General syntax:

The query string of the URL needs to be [URLEncoded](#), [URL Encode Page](#).

```
<a href="http[s]://<server_name>[:server_port]/sdweb/<load_servlet>?
<parameter_name>=<parameter_value>[&<parameter_name>=<parameter_value>]">
```

Request parameters

The loading process can be controlled by different request parameters.

- Define the document ID
- Compose a unique document ID
- Define the DMS plugin
- Define the Signature Archive plugin
- Define the result URL

Define the document ID

Parameter: docid (optional)

Defines a unique ID for the document to be loaded. The docid part of the URL when the document is shown in the browser (last part of the URL).

Example

`http://localhost:6610/sdweb/signdoc/showgui/1239183368386`

Optional parameter: If not specified, the server will use the current timestamp (e.g. 2014-10-23_10-00-08-891) as ID.

Syntax of parameter value: A simple string value without spaces and special characters. It has to match the following regular expression `[\w-\.]`* which means that it can consist of the characters: 0-9, a-Z, -, _ .

Valid Examples: `docid=a_document_id`, `docid=0123456789`, `docid=docname_1234`.

Compose a unique document ID

Parameter: `docidsalt` (optional)

Additional information for composing a unique document ID in the IDms plugin.

Optional parameter: If not specified, the server will pass a timestamp to the IDms plugin.

Syntax of parameter value: A simple string value without spaces and special characters. It has to match the following regular expression `[\w-\.]`* which means that it can consist of the characters: 0-9, a-Z, -, _ .

Valid Examples: `123454`, `customer_contract`, ...

Define the DMS plugin

Parameter: `dmsid` (optional)

Defines the DMS plugin to be used to handle the document. The `dmsid` corresponds to a loaded IDms plugin in SignDoc Web (e.g. `de.softpro.sdweb.plugins.impl.FileDms`).

Optional parameter: If not specified, the server will not use an DMS plugin. The document cannot be stored on backend side in this case.


Syntax of parameter value: A simple string value without spaces and special characters. It has to match the following regular expression `[\w-\.]`* which means that it can consist of the characters: 0-9, a-Z, -, _ .

Valid Examples: `de.softpro.sdweb.plugins.impl.FileDms`

Define the Signature Archive plugin

Parameter: `signaturearchiveid` (optional)

Defines the Signature Archive plugin to be used to handle the document signatures. The `signaturearchiveid` corresponds to a loaded ISignatureArchive plugin in SignDoc Web (e.g. `de.softpro.sdweb.plugins.impl.SignArchive`).

 This parameter has to be set if validation or storage for signatures should be used!

Optional parameter: If not specified, the server will not use a Signature Archive plugin. The signatures cannot be stored or validated on backend side in this case.

Syntax of parameter value: A simple string value without spaces and special characters. It has to match the following regular expression `[\w-\.]`* which means that it can consist of the characters: 0-9, a-Z, -, _ .

Valid Examples: `de.softpro.sdweb.plugins.impl.SignArchive`

Define the result URL

Parameter: resulturl (optional)

The URL to redirect the browser window to after finishing the signing process. A valid Fully qualified URL. When passes as Get parameter, the URL has to be in URL Encoded format [URL Encoded](#), [URL Encode Page](#).

Optional parameter: If not specified, the browser will show an internal result page.

SignDoc Web attaches the following parameters:

Parameter Name	Parameter Value	Description
sdweb_docid	document	The document ID of the processed document.
sdweb_result	success	Set when the process completed successfully.
sdweb_result	cancel	Set when the process was cancelled by the user.
sdweb_result	error	Set when there was an error.

Example

Defining Result URL

`http://www.softpro.de`

will redirect to

`http://www.softpro.de?sdweb_docid=1243946562111&sdweb_result=success`

Create or update a field in the document

Parameter: cmd

Syntax of request parameter cmd:

`name="cmd[_<uniqueid e.g. number>]"`

Examples

`name="cmd_1", name="cmd_someid"`

This request parameter is used for setting metadata and inserting or updating form elements or signature fields. When multiple fields should be updated in one step, the request parameter must to be extended with "`_<unique_id>`" (1 based and ascending number is recommended), so that the parameter names are unique.

Add text to a document

It's possible to add text to a document page.

i Text can be added only to PDF documents.

```
value="text=<the_text>|page=<page_number>|searchtext=<text_to_search>|
left=<left_coordinate>|bottom=<bottom_coordinate>|fontname=<fontname>|
fontsize=<font_size_in_pt>|textcolor=<text_color>|opacity=<opacity>|
offsetx=<offsetx>|offsety=<offsety>"
```

Examples

Add the semitransparent text "Lorem ipsum" on page 1 with the coordinates (10/10) and the fontsize 100pt using the standard Font Helvetica.

```
value="type=addtext|text=Lorem ipsum|pages=1|left=10|bottom=10|fontsize=100|
fontname=Helvetica|textcolor=#FF0000|opacity=0.5"
```

Add the semitransparent text "Lorem ipsum" on all pages (-1) with the coordinates (10/10) and the fontsize 100pt using the standard Font Helvetica.

```
value="type=addtext|text=Lorem ipsum|pages=-1|left=10|bottom=10|fontsize=100|
fontname=Helvetica|textcolor=#FF0000|opacity=0.5"
```

Add the semitransparent text "Lorem ipsum" on page 1,2 and 4 with the coordinates (10/10) and the fontsize 100pt using the standard Font Helvetica.

```
value="type=addtext|text=Lorem ipsum|pages=1,2,4|left=10|bottom=10|
fontsize=100|fontname=Helvetica|textcolor=#FF0000|opacity=0.5"
```

Parameters

Inline Parameter	Value/Syntax/Description	Remarks
text	The text to add.	mandatory
type	addtext	mandatory
pages	The pages where to add the text. Examples: "1" (page 1); "1,2,4" (pages 1,2 and 4); "-1" (all pages)	mandatory, if coordinates "left" and "bottom" are undefined
searchtext	The found positions determine the coordinates where to add the text.	mandatory, if coordinates "left" and "bottom" are undefined
offsetx	Offset in horizontal direction. Positive values means "right".	optional
left (or x)	Offset in vertical direction. Positive values means "up".	optional
bottom (or y)	Bottom coordinate where to add the text (origin is lower left corner).	mandatory, if "searchtext" undefined

Inline Parameter	Value/Syntax/Description	Remarks
fontname	The fontname. PDF Standard Fonts (e.g. Helvetica) can be used without extra configuration. Using TTF Fonts requires a font mapping configuration. For more information see <i>Kofax SignDoc Web Administrator's Guide</i> , chapter "Font mapping configuration".	optional (default: Helvetica)
fontsize	The fontsize in pt.	optional (default: 12)
textcolor	The color of the text in HTML RGB syntax.	optional (default: #000000)
opacity	The opacity of the text. Range: 0.0 (transparent) - 1.0 (opaque).	optional (default: 1.0)

Insert a form field by coordinates

A form field is inserted by defining exact document coordinates.

```
value="name=<field_name>|page=<page_number>|bottom=<bottom_coordinate>|  
left=<left_coordinate>|width=<width>|height=<height>|type=formfield|  
subtype=<subtype>|required=<true|false>|readonly=<true|false>|label=<label>"
```

Example

Creating a signature field sig1 on Page 1 with the coordinates (10/10/150/50).

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

Inline Parameter	Value/Syntax/Description	Remarks
name	The name of the field.	mandatory
page	The page number, 0<value<pagecount.	mandatory
bottom,left,width,height	Document coordinates (origin in the lower-left corner).	mandatory
type	formfield	mandatory
subtype	capture signature c2s image_capture checkbox textfield	mandatory
required	true or false, sets the "required" attribute.	optional
readonly	true or false, read only/locked field (not useful with signature fields).	optional

Inline Parameter	Value/Syntax/Description	Remarks
label	Sets a friendly name (label) for the field.	optional

Insert a form field with position located by text phrase

A form field is inserted by locating a text phrase in the document as anchor and placing the signature field with a relative offset.

Note Depending on the format of the PDF, it might be not possible to find all/any text-strings, e.g. the page content is an embedded image.

To use unique text phrases, it is possible to have this phrase in the document as white text on white background, so it is not visible when printing.

```
value="name=<field_name>|page=<page_number>|width=<width>|
height=<height>|type=formfield|subtype=signature|searchtext=<search_text>|
searchpages=<page_list>|offsetx=<offsetx>|offsety=<offsety>|required=<true|
false>|readonly=<true|false>|label=<label>"
```

Example

Insert a new signature field in all places where the text "Customer_Signature" is found in the document. The signature field is placed with a relative (x/y) offset of (-45/30) and has the dimension of 140/50 (all in document coordinates). The signature field is marked as required and has to be signed.

```
value="name=SignatureAuto|searchtext=Customer_Signature|width=140|height=50|
offsetx=-45|offsety=30|type=formfield|subtype=signature|required=true"
```

Parameter	Value/Syntax/Description	Remarks
name	The field name.	mandatory
page	The page number, 0<value<pagecount.	optional
width,height	Document coordinates (origin in the lower-left corner).	mandatory
type	formfield	mandatory
subtype	capture signature c2s image_capture checkbox textfield	mandatory
searchtext	A text phrase to search in the document.	mandatory
searchpages	A comma separated list of pages to look for the searchtext.	optional, if not specified, search all pages
offsetx	Offset in horizontal direction positive values means right.	optional
offsety	Offset in vertical direction positive values means up.	optional

Parameter	Value/Syntax/Description	Remarks
required	true or false, sets the "required" attribute	optional
readonly	true or false, read only/locked field (not useful with signature fields)	optional
label	Sets a friendly name (label) for the field.	optional

Insert a general capture field

Instead of inserting a specific capture field, like a signature or an image_capture field, it is also possible to insert a general capture field without appointing to a constant capture method. With a general capture field the user can decide by what means he captures a document attribute. Via command interface it is possible to insert a form field with subtype 'capture' and a choice list (parameter capturechoice) with the capture methods which should be available later for the user.

```
value="name=<field_name>|page=<page_number>|<location parameters>|
type=formfield|subtype=capture|required=<true|false>|readonly=<true|false>|
label=<label>|capturechoice=<signature|c2s|image_capture>"
```

Example

Creating a capture field capture1 on Page 1 with some coordinates and a choice list for capturing either an image from a connected camera or Click-to-Sign signature.

```
value="name=capture1|page=1|type=formfield|subtype=capture|bottom=100|
left=100|width=150|height=50|capturechoice=image_capture,c2s"
```

Inline Parameter	Value/Syntax/Description	Remarks
name	The field name.	mandatory
page	The page number, 0<value<pagecount.	optional
<location parameters>	See Insert a form field by coordinates or by text-phrase	mandatory
type	formfield	mandatory
subtype	capture	mandatory
capturechoice	signature c2s image_capture	optional
<common attributes>	like 'required', 'readonly' or 'label'. See Insert a form field by coordinates or by text-phrase	optional

i If 'capturechoice' is empty or omitted a default choice ['signature', 'image_capture', 'c2s'] is offered to the user.

The default choice can be overwritten in sdweb_config.groovy with the entry

```
sdweb.capture.subtype.choice=[...]
```

See also *Kofax SignDoc Web Administrator's Guide*, chapter "Configuration file sdweb_config.groovy" for detailed information on options.

Update the value and attribute of an existing form field or signature field

Several attributes and the value of an existing form field can be updated.

```
value="name=<field_name>|page=<page_number>|bottom=<bottom_coordinate>|
left=<left_coordinate>|width=<width>|height=<height>|type=formfield|
subtype=<textfield|signature|checkbox>|required=<true|false>|readonly=<true|
false>|label=<label>|value=<fieldvalue>"
```

Example

Prefilling an existing text field with the name Texteingabe7 with the Text "Test".

```
value="name=Texteingabe7|type=formfield|value=Test|subtype=textfield"
```

Checking a check box with the name "Check Box3".

```
value="name=Check Box3|type=formfield|value=true|subtype=checkbox"
```

i The subtype of form field can only be changed, if the target subtype is either signature, image_capture, c2s or capture.

Inline Parameter	Value/Syntax/Description	Remarks
name	The existing field name.	mandatory
page	The page number, 0<value<pagecount.	mandatory, updatable
bottom,left,width,height	Document coordinates (origin in the lower-left corner).	optional, updatable
type	formfield	mandatory
subtype	textfield or signature or checkbox	mandatory, updatable
required	true or false, sets the "required" attribute	optional, updatable
readonly	true or false, read only/locked field (not useful with signature fields)	optional, updatable
maxlength (type==formfield subtype=textfield)	Maximum characters for a textfield's content, a number > 0	optional, updatable
label	Sets a friendly name (label) for the field.	optional, updatable
value (type==formfield subtype=textfield)	A string value of the textfield to set.	mandatory for subtype=textfield
value (type==formfield subtype=checkbox)	true or false	mandatory for subtype=checkbox

Inline Parameter	Value/Syntax/Description	Remarks
value (type==formfield subtype=signature)	not applicable	optional, ignored
validpattern (type==formfield subtype=textfield)	<p>Validation pattern (regular expression) for text field value verification. The value of the entered text must match this pattern otherwise it will be rejected.</p> <p>Note The regular expression pattern must be URL-encoded!</p> <p>Example</p> <p>The email pattern</p> <p><code>^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}\$</code></p> <p>must be encoded as</p> <p><code>%5E%5Ba-zA-Z0-9._-%5D%2B%40%5Ba-zA-Z0-9._-%5D%2B%5C.%5Ba-zA-Z%5D%7B2%2C6%7D%24</code></p>	<p>Optional parameter.</p> <p>Info to URL encoding can be found under:</p> <p>http://www.w3schools.com/TAGs/ref_urlencode.asp</p>
validmessage (type==formfield subtype=textfield)	<p>Validation message will be displayed by SignDoc Web client if entered text does not match the validation pattern.</p> <p>Example</p> <p>An email address such as John.Smith@example.com is made up of a local part, an @ symbol, then a domain part.</p>	optional parameter.

Convert an existing form field to a signature field

Convert an existing text field to a signature field with the same position and the same dimensions:

```
value="name=<field_name>|type=formfield|subtype=signature|required=<true|false>|label=<label>"
```

Example

Converting the form field Texteingabe7 to a required signature field:

```
value="name=Texteingabe7|type=formfield|subtype=signature|required=true"
```

Inline Parameter	Value/Syntax/Description	Remarks
name	The existing field name.	mandatory
type	formfield	mandatory
subtype	signature	mandatory, updatable
required	true or false, sets the "required" attribute	optional, updatable

Inline Parameter	Value/Syntax/Description	Remarks
label	Sets a friendly name (label) for the field.	optional, updatable

Set metadata

Metadata can be stored in the documents, that is available for the different plugins, e.g. DMS plugin. It can be used to store additional customer specific data as key/value pairs inside the document.

For PDF document, XMP is used for storing the metadata.

If metadata should be displayable with any tools you have to store it as 'public'.

By default metadata is stored in 'private' properties which are not readable by any standard tools.

Syntax 1

Setting metadata in 'private' area of the document with

```
sdweb://command/metadata/add/<field_name>?<field_value>
```

Setting metadata as 'public' property in the document with

```
sdweb://command/metadata/add_public/<field_name>?<field_value>
```

Example

Setting metadata customer_id to CID123456789 in 'private' properties:

```
value="sdweb://command/metadata/add/customer_id?CID123456789"
```

Setting metadata classification to URGENT in 'public' properties:

```
value="sdweb://command/metadata/add_public/classfication?URGENT"
```

Inline Parameter	Value/Syntax/Description	Remarks
metadata_name	The metadata name.	mandatory
metadata_value	The metadata value as string.	mandatory

Syntax 2

```
value="name=<metadata_name>|type=metadata|value=<metadata_value>[|  
decode=base64][|visibility=<visibility_value>]"
```

Example

Setting metadata customer_id to CID123456789 in 'private' properties:

```
value="name=customer_id|type=metadata|value=CID123456789|visibility=private"
```

Setting metadata classification to URGENT in 'public' properties:


```
value="name=classification|type=metadata|value=URGENT|visibility=public"
```

Inline Parameter	Value/Syntax/Description	Remarks
metadata_name	The metadata name.	mandatory
type	metadata	mandatory
metadata_value	The metadata value as string.	mandatory
decode	Possible values: base64. If specified, the data will be decoded.	optional (default:plain)
visibility_value	Possible values: private or public	optional (default: private)

Signature archive interaction

A signature can be forced to validate (match) against a reference signature (e.g. of FraudOne), before it is accepted as signature. A signature can be stored after capturing a signature database (e.g. in FraudOne).

i SignDoc Web must have a SignatureArchive plugin installed and loaded to provide this functionality:

```
value="name=<field_name>|type=signaturearchive|subtype=<validate|store>|  
value=<signerid>"
```

Example

Setting the signature field signature1 to be validated before signing:

```
value="name=signature1|type=signaturearchive|subtype=validate|value=chi"
```

Setting the signature field signature1 to be stored after signing:

```
value="name=signature1|type=signaturearchive|subtype=store|value=chi"
```

Inline Parameter	Value/Syntax/Description	Remarks
name	The field name.	mandatory
type	signaturearchive	mandatory
subtype	validate or store	mandatory
value	The signerid to use (format is plugin-specific).	mandatory

Field validation before update

If any field is updated in Signdoc Web it is possible to validate this before with the help of a customer-specific validation plugin which implements the IDocumentValidator interface.

The `validateFieldChange()` method of this plugin is called directly before the update action is performed on the server (see [SignDoc Web plugins general information](#) for general plugin handling in SignDoc Web).

Method declaration:

```
Map<IDocumentField, String> validateFieldChange(IDocumentData documentData,  
IDocumentField documentField, Locale locale) throws PluginException;
```

The main task of the `validateFieldChange()` method is to decide whether the field update should be performed on the server or not.

The method returns a map (`Map<IDocumentField, String>`) with one or more `IDocumentField` entries (key) each with an appropriate explanation (value) if the change is not acceptable.

These explanations are returned together with the respective field name to the client for display.

In this case the field update is not performed!



Field change during validation



In the `validateFieldChange()` method it is also possible to change some specific parts of one or more fields which are reflected in the client. The parameters of this method contains the implemented objects of `IDocumentData` and `IDocumentField`. The `IDocumentField` contains information about the current field which should be updated.

The current field can be changed by using the setter methods of the `IDocumentField` object.

The `IDocumentData` object allows amongst others access to all fields (`getFields` method returns map with `IDocumentFields` objects) and to the `MetaData` (`getMetaData` method returns map with `IMetadataEntry` objects) in the document. These fields can then be changed in the same matter via setter methods of the `IDocumentField` object.


Since the `IDocumentField` objects could represent different field types the according values (`get/setValue()`) have also different meanings. The value is a Java object which could be from type string or boolean.

Field Type	Description
Text field	The string value of the text field. It can be changed in the <code>validateFieldChange()</code> method.
Check Box	The boolean value as string ("true" or "false") indicates whether the checkbox is checked or not. A Check Box state can be changed by setting the value to (String) "true" or "false".
Signature	The boolean value indicates whether a signature is captured in the signature field or not. <div> A change of this value is not considered!</div>
Image	The boolean value indicates whether the Image is captured or not. <div> A change of this value is not considered!</div>

Field Type	Description
C2S	<p>The boolean value indicates whether the Click-to-Sign signature is captured or not.</p> <p> A change of this value is not considered!</p>
Radio Button Group	<p>The string value contains the currently selected Radio Button value (or 'off' if nothing is selected).</p> <p>A Radio Button selection can be changed by setting the value of the Radio Button Group to another (existing) Radio Button value.</p> <p> Individual Radio Buttons (widgets) are not included in the list of IDocumentFields within IDocumentData.</p>

A Metadata value also can be changed with the `setStringValue()` method of the `IMetadataEntry`.

It is also possible to create a new `IMetadataEntry` object which can be added to the map that is returned by the `getMetaData()` method of `IDocumentData`. A changed or added `MetaData` entry is not passed to the client directly after the update but it will be stored in the document if it is archived.

 Any changes will be only considered if the returned map (with rejected `IDocumentField` objects) is empty or null.

In order to avoid that the `validateFieldChange()` method is called for each field update it is possible to configure in `sdweb_config.groovy` (setting: `sdweb.validate.before_update.fieldtypes`) for which kind of form field type the validation is necessary. See also *Kofax SignDoc Web Administrator's Guide*, chapter "Configuration file `sdweb_config.groovy`" for more information.

By default the plugin method is not called before an update is requested.

If only one or only some specific fields must be validated it is advisable to mark a field via command interface that it must be validated before updated.

Inline Parameter	Values, Syntax, Description	Values, Syntax, Description
<code>validate_update</code>	<p>Value type: Boolean (default: false).</p> <p>Specifies whether the <code>validateFieldChange()</code> method of the <code>IDocumentValidator</code> interface implementation will be called before updating the field on SignDoc Web server.</p>	Optional

Example

```
<input name="cmd_1" value="name=Customer_Name|page=1|type=formfield|
subtype=textfield|validate_update=true"/>
```

See also [Create or update a field in the document](#). An additional performance log entry is generated for the "Validation before field update" action within the interactive phase '45-01'='IP-FIELD-VALIDATE-UPDATE'. See also *Kofax SignDoc Web Administrator's Guide*, chapter "Logging".

Signature watermark

By default a captured signature is rendered in a signature field on a white background, but it is also possible to define a customer specific background image, a so-called signature watermark. If a signature watermark is enabled the appearance of the signature in the document view is changed but not the captured signature itself.

Example



In general you can set watermark options either generally in `sdweb_config.groovy` or individually per signature field in a command during loading of the document.

The individual settings have more priority than the global configured settings for the watermark.

See *Kofax SignDoc Web Administrator's Guide*, chapter "Configuration file `sdweb_config.groovy`" for a description of the global settings.

The individual watermark options can be appended to any other signature field commands.

The default values in the parameter value description are only valid if nothing else is specified in `sdweb_config.groovy`.

The watermark display is limited by an invisible area, the watermark frame.

For the watermark view within the frame you can define a scale option.

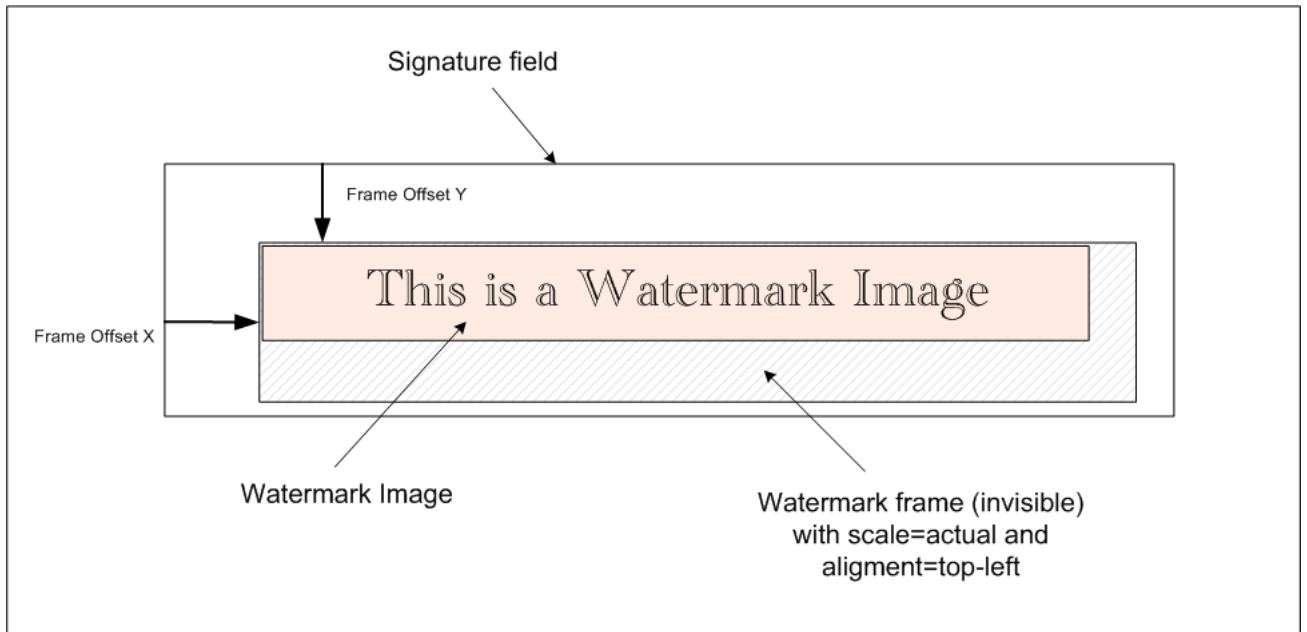
The watermark can be inserted into the frame as it is (unchanged size) with scale option 'actual'.

The image can be scaled to the frame size with scale option 'fit' or 'stretch'.

If the watermark image does not fill the frame completely (possible with scale option 'actual' and 'fit') it can be adjusted within the frame with an alignment configuration, e.g. 'middle-center'.

If the watermark image is bigger than the frame (possible with scale option 'actual') it will be cropped in order to fit into the frame.

The watermark frame (with the watermark image) itself is positioned into the signature field according to the configurable frame offsets. If no frame offsets are defined the frame is placed also in to the signature field according to the defined alignment.



Inline Parameter	Values, Syntax, Description	Remarks
watermark_enable	Value type: boolean (default: false). Specifies whether a watermark image is placed behind a captured signature in the signature field. If set to true, SignDoc Web will try to add a configured watermark behind the signature.	optional
watermark_image_template	Value type: string (default: watermark.bmp). The name of the watermark image which should be placed as background image for a captured signature (for the document view).	optional

Inline Parameter	Values, Syntax, Description	Remarks
watermark_image_scale	<p>Value type: string (default: actual).</p> <p>Specifies the scaling strategy of the watermark image within the watermark frame. The watermark frame is the area where the watermark image is adjusted into. The watermark frame itself is adjusted then into the signature field, either by frame offset coordinates (sdweb.signature.watermark.frame.offset_x and sdweb.signature.watermark.frame.offset_y) or alternatively according the alignment setting (sdweb.signature.watermark.image.alignment).</p> <ul style="list-style-type: none"> • "actual" will use the actual size of the watermark image when it is positioned in the watermark frame. • "fit" will scale the image proportionately to the size of the watermark frame. • "stretch" will stretch the image disproportionately to fit the entire frame. 	optional
watermark_image_alignment	<p>Value type: string, "middle-center"</p> <p>The watermark image alignment is only applicable if width and/or height of the watermark image is smaller than the watermark frame and the scale is set to "actual". If scale is set to "fit", it depends on whether the frame is longer than the fitted image (horizontal alignment possible) or taller than the fitted image (vertical alignment possible). Scale option "stretch" will not have any impact since it is stretched to fit the frame.</p> <p>Possible alignment settings are "top-left", "top-center", "top-right", "middle-left", "middle-center", "middle-right", "bottom-left", "bottom-center" and "bottom-right".</p>	optional
watermark_frame_offset_x	<p>Value type: Integer, -1</p> <p>Horizontal offset (in pixels) between top-left corner of the signature field and the top-left corner of the watermark frame . Positive value means top-left corner of the watermark frame is on the right of the top-left corner of the signature field. A negative value means that both offsets (x and y) are not used for frame positioning within the signature field. In this case the sdweb.signature.watermark.image.alignment setting is used.</p> <p>The watermark image alignment is only applicable then if width and/or height of the frame is smaller than the signature field.</p>	optional

Inline Parameter	Values, Syntax, Description	Remarks
watermark_frame_offset_y	Value type: Integer, -1 Vertical offset (in pixels) between top-left corner of the signature field and the top-left corner of the watermark frame. Positive value means top-left corner of the watermark frame is below the top-left corner of the signature field. A negative value means that both offsets (x and y) are not used for frame positioning within the signature field. In this case the sdweb.signature.watermark.image.alignment setting is used. The watermark image alignment is only applicable then if width and/or height of the frame is smaller than the signature field.	optional
watermark_frame_width	Value type: Integer, -1 Width of the watermark frame (if the value is not negative). A negative value means that the watermark frame has the same width as the signature field.	optional
watermark_frame_height	Value type: Integer, -1 Height of the watermark frame (if the value is not negative). A negative value means that the watermark frame has the same height as the signature field.	optional
watermark_image_opacity	Value type: Integer, 100 Specifies the watermark image opacity in percent. The value must be between 0 and 100. 100(%) means that the watermark image is completely opaque. 0 would make the watermark invisible because it is completely transparent. This setting can be useful to show the signature more prominent compared to the watermark image which is behind the signature image.	optional

Insert a new signature field on Page 1 with the coordinates x=100, Y=100 (with origin lower left corner) with width=350 and height=100.

The watermark image watermark.bmp should be placed into a watermark frame with 330 pixel width and 90 pixel height without scaling (scale=actual) and aligned to the bottom right corner (be it that the watermark image is smaller than the frame) of the frame.

The watermark frame itself is positioned in the left upper corner of the signature field with an offset of x=10 and y=10.

```
value="name=Signature1|page=1|type=formfield|subtype=signature|bottom=100|left=100|width=350|height=100|watermark_enable=true|watermark_image_template=watermark.bmp|watermark_image_scale=actual|watermark_image_alignment=bottom-right|watermark_frame_offset_x=10|watermark_frame_offset_y=10|watermark_frame_width=330|watermark_frame_height=90|watermark_image_opacity=50"
```

Remove an existing form field or signature field

A field is referenced by name. Also logical names are supported.

A logical name identifies a field which could occur several times on different pages.

A list of pages can be defined, in order to determine on which page(s) the field (referenced by the logical name) should be deleted.

```
value="name=<field_name>|pages=<page_numbers>"
```

Example 1

Remove (signature) field with name SPFID_CUST_SIGNATURE_SPFID_2 in the requested document:

```
<input type="hidden" name="cmd1" value="type=removefield|
name=SPFID_CUST_SIGNATURE_SPFID_2"/>
```

Example 2

Remove field with name Customer_Number in the pages 1,2 and 4:

```
<input type="hidden" name="cmd1" value="type=removefield|name=Customer_Number|
pages=1,2,4"/>
```

Inline Parameter	Value/Syntax/Description
name	The existing field name
pages	Examples "1" (page 1); "1,2,4" (pages 1,2 and 4); "-1" (all pages)

i The remove action fails if the field is set to readonly. Read only fields can be removed if `sdweb.cmd.allow.update.readonly.editfields=true` is set in `sdweb_config.groovy`.

SignDoc Web field locking

Locked fields in SignDoc Web are PDF fields with read only attribute.

The read only flag can be set directly via servlet command for creating or updating a field in the document starting with 'cmd' (or old syntax 'createorupdate').

Example for opening a (template) document and adding a new signature field:

```
<form id="the_form" action="http://localhost:6610/sdweb/load/bytemplate"
target="_blank" method="post">
<input type="submit" name="opendoc" value="open document"/>
<input type="hidden" name="template" value="softpro_banking_trapeza_en.pdf" size="80"/>
<input type="hidden" name="docid" value="TestSignature" />
<input type="hidden" name="cmd_1" value="name=Approver|page=1|type=formfield|
subtype=signature|bottom=100|left=10|width=150|height=50|lock_after_sign=self" />
</form>
```


Deferred locking of fields (setting of read only attributes) can be requested for signed signature fields with 3 different signature field attributes.

```
lock_after_sign={self | all | all_editfields | all_editfields_not_self |  
fieldname[,fieldname...] }
```

```
lock_after_sign_exclude=fieldname[,fieldname...]
```

```
archive_action=lock_all_if_signed
```

lock_after_sign

The attribute value 'self' means, that the signature field itself is locked after it is signed.

All changeable fields (including signature fields) are locked after signing the signature field if value is set to 'all'.

With value 'all_editfields' only the edit fields (text input fields, check boxes and radio buttons) and the signed signature field itself are locked if the appropriate signature is signed.

The lock attribute can be appended to the other signature field attributes in the command, e.g.

```
<input type="hidden" name="cmd_1" value="name=Approver|page=1|type=formfield|  
subtype=signature|bottom=100|left=10|width=150|height=50|lock_after_sign=self" />
```


The flag 'all_editfields_not_self' locks also only the edit fields but without locking the signed signature field itself.

If you want to define only specific fields which should be locked after signing the signature field you can append the (real PDF) field names as value for the attribute lock_after_sign, e.g.

```
<input type="hidden" name="cmd_1" value="name=Approver|page=1|  
type=formfield|subtype=signature|bottom=100|left=10|width=150|height=50|  
lock_after_sign=field1,field2,field3"/>
```

The field names that has to be locked after signing a specific signature field must be separated with a comma (in case of several field names).

The field names must not match with the reserved words (self, all, all_editfields and all_editfields_not_self).

 The field list separator can be changed in sdweb_config.groovy with entry sdweb.lockfields.list_separator="separator character", whereas "," is the default.

lock_after_sign_exclude

The lock_after_sign_exclude attribute is followed by a field name or a list of field names (separator, see lock_after_sign) which should be excluded from locking.

All changeable fields except the specified fields are set to read only if the signature is signed, e.g.


```
<input type="hidden" name="cmd_1" value="name=Approver|page=1|  
type=formfield|subtype=signature|bottom=100|left=10|width=150|height=50|  
lock_after_sign_exclude=field1,field2,field3" />
```

archive_action

The archive_action attribute has only one valid value lock_all_if_signed.

This attribute setting locks all fields after signing not till the document is archived. That means you can make any changes in the document also after signing the signature field but only until archiving.

This makes sense if you want to prevent any changes after signing but independent from the sequence of the changes in the document (including signing), at least until archiving.

 archive_action=lock_all_if_signed can be combined with one of the other lock attributes.

```
<input type="hidden" name="cmd_1" value="name=Approver|page=1|type=formfield|
subtype=signature|bottom=100|left=10|width=150|height=50|lock_after_sign=self|
archive_action=lock_all_if_signed"/>
```

URI syntax

Another URI-based syntax is possible in the commands in order to lock fields for a signed signature field.

It can only be used for setting lock attributes for a signature field but not for any other settings as with the other already described syntax.

Syntax

```
sdweb://command/lockfields/signature/{include|exclude|all|none}/signature
field name?[fieldname[|fieldname... ]]
```


The lock commands followed by the prefix:

```
sdweb://command/lockfields/signature/
```

cause lock actions for the specified signature field name.

Command	Description
all	All fields are locked if a signature is captured for the named signature field. Example Result All changeable fields in the document are locked after signing signature field sig3.
none	No field is locked if a signature is captured for the named signature field. Example sdweb://command/lockfields/signature/none/sig4? Result This can be used for reset any previously defined lock specific settings for a signature field.

Command	Description
include	All appended fields are locked if the signature is captured for the named signature field Example <code>sdweb://command/lockfields/signature/include/sig1?field1 field2 field3</code> Result field1, field2 and field3 are locked after signature field sig1 is signed
exclude	All fields are locked if a signature is captured for the named signature field except the fields in the subsequent field names list. Example <code>sdweb://command/lockfields/signature/exclude/sig2?field2 field5 field6</code> Result Only field2, field5 and field6 are not locked after signing signature field sig2.

 The field separator for the URI syntax can be changed via configuration entry:

```
sdweb.lockfields.list_separator_uri=... (default value: "\\|" for |)
```

Locked fields are set to read only. Fields with flag read only should not be updatable by the user with the GUI.

But editable fields (checkbox or text) can be also changed via described servlet interface.

With this "command" interface it could be wanted that a field can be updated by a program although it is marked as read only, e.g. for preallocation of (actually) readonly fields like the customer or account name.

Since it is not always clear how to handle read only fields, the configuration setting `sdweb.cmd.allow.update.readonly.editfields` (in `sdweb_config.groovy`) determines whether update of read only fields is allowed via "command" servlet interface or not.

By default, it is allowed (as it was before implementing the setting) to update read only fields.

With

```
sdweb.cmd.allow.update.readonly.editfields=false
```

in `sdweb_config.groovy` this setting can be overwritten.

If anybody (or better any program) tries then to change the value of a read only field the update will not be performed.

Only a log entry (log level info) will be written with the note, that the field is read only and cannot be changed

(e.g. "field READONLY_FIELD is read only (locked) and cannot be changed").

Flatten locked fields

Locked fields are set to read only by default.

With `sdweb.config.groovy` configuration entry

```
sdweb.locking.flatten=true
```

all read only fields are flattened.

Flattening of fields means that they will be removed and cannot be accessed any more. The appearance of the field is kept which means that all signatures, all text input and all other settings are still visible but cannot be changed.

The default is

```
sdweb.locking.flatten=false
```

Document metadata

Metadata	Description
SIGNDOCWEB_SIMPLE_SIGNING_WORKFLOW_LIST	<p>A comma separated list of signature-field-names (realid) that will be used, if the setting SigningWorkflow is enabled.</p> <p>For details how to use the SignignWorkflow consult <code>signdoc_configuration.xml</code> or <code>mobile_configuration.xml</code>.</p> <p>See <i>Kofax SignDoc Web Administrator's Guide</i>, chapter "Configurable toolbar".</p>

Signature and form fields

Define an existing signature field as a required/mandatory field

When opening the document, add a `createorupdate` statement, that contains `required=true` as inline parameter.

Example form request:

```
<form action="http://localhost:6610/sdweb/load/bytemplate" name="bytemplate"
target="_blank" enctype="multipart/form-data" method="post">
  <input type="hidden" name="dmsid" value="de.softpro.sdweb.plugins.impl.FileDms"/>
  <input type="hidden" name="template" value="trapeza_bank_apac_account_opening.pdf"/>
  <input type="hidden" name="createorupdate_1001"
value="name=topmostSubform[0].Page1[0].Signature1_001[0]|type=formfield|
subtype=signature|required=true|tooltip=Customer Signature 1...|friendlyname=Customer
Signature 1" />
  <input type="hidden" name="createorupdate_1002"
value="name=topmostSubform[0].Page1[0].Signature2_001[0]|type=formfield|
subtype=signature|required=true|tooltip=Customer Signature 2...|friendlyname=Customer
Signature 2" />
  <input type="submit" name="opendoc" value="open"/>
</form>
```

The same example using the URL query string:

```
<a href="http://localhost:6610/sdweb/load/bytemplate?
```

```
dmsid=de.softpro.sdweb.plugins.impl.FileDms&template=
trapeza_bank_apac_account_opening.pdf&createorupdate_1001=
name=topmostSubform[0].Page1[0].Signature1_001[0]|type=formfield|subtype=signature|
required=true|tooltip=Customer Signature 1...|friendlyname=Customer Signature
1&createorupdate_1002=name=topmostSubform[0].Page1[0].Signature2_001[0]|
type=formfield|subtype=signature|required=true|tooltip=Customer Signature 2...|
friendlyname=Customer Signature 2"
>Open the document</a>
```

Add a label and/or tooltip to a form or signature field

When opening the document, add a 'createorupdate' statement, that contains friendlyname=<a label> and/or tooltip=<the tooltip> as inline parameter.

Example form request:

```
<form action="http://localhost:6610/sdweb/load/bytemplate" name="bytemplate"
target="_blank" enctype="multipart/form-data" method="post">
  <input type="hidden" name="dmsid" value="de.softpro.sdweb.plugins.impl.FileDms"/>
  <input type="hidden" name="template" value="trapeza_bank_apac_account_opening.pdf"/>
  <input type="hidden" name="createorupdate_1001"
value="name=topmostSubform[0].Page1[0].Signature1_001[0]|type=formfield|
subtype=signature|required=true|tooltip=Customer Signature 1...|friendlyname=Customer
Signature 1" />
  <input type="hidden" name="createorupdate_1002"
value="name=topmostSubform[0].Page1[0].Signature2_001[0]|type=formfield|
subtype=signature|required=true|tooltip=Customer Signature 2...|friendlyname=Customer
Signature 2" />
  <input type="submit" name="opendoc" value="open"/>
</form>
```

The same example using the URL query string:

```
<a href="http://localhost:6610/sdweb/load/bytemplate?
dmsid=de.softpro.sdweb.plugins.impl.FileDms&template=
trapeza_bank_apac_account_opening.pdf&createorupdate_1001=
name=topmostSubform[0].Page1[0].Signature1_001[0]|type=formfield|subtype=signature|
required=true|tooltip=Customer Signature 1...|friendlyname=Customer Signature
1&createorupdate_1002=name=topmostSubform[0].Page1[0].Signature2_001[0]|
type=formfield|subtype=signature|required=true|tooltip=Customer Signature 2...|
friendlyname=Customer Signature 2"
>Open the document</a>
```

Radio buttons

Description

In this section radio button support in PDF documents is described. Radio buttons can be selected by SignDoc Web browser clients as well as by supported Mobile Clients. Radio buttons can be created or changed in SignDoc Web Server for a PDF document via command interface.

Radio buttons are arranged in groups of two or more and displayed on screen. A radio button group can have several radio buttons, but a single radio button is assigned to exactly one group. When the user selects a radio button, any previously selected radio button in the same group becomes deselected. Selecting a radio button is done by clicking the mouse on the button. It is possible that initially none of the radio buttons in a group is selected. This state cannot be restored by interacting with the radio button widget (but it is possible through SignDoc Web command interface).

A field can be created or updated in the document via cmd request parameter. Find a description of the command interface to SignDoc Web under [Integration in existing web applications](#).

Usage

```
name=<field_name>|type=formfield|subtype=radiobutton|buttonid=<buttonid>|value=<true|false>|page=<page_number>|bottom=<bottom_coordinate>|left=<left_coordinate>|width=<width>|height=<height>|required=<true|false>|readonly=<true|false>|label=<label>
```

Inserting radio buttons by coordinates

A radio button is inserted by defining exact document coordinates.

Example

Create two radio buttons on page 1.

The radio button group name "gender" is defined for both radio buttons with the name attribute name=gender.

The first radio button with buttonid=female is selected (value=true) and placed at the coordinates bottom=50, left=200, width=10, height=10 (origin lower left corner).

The second radio button with buttonid=male is not selected (value=false) and placed at the coordinates bottom=30, left=200, width=10, height=10.

```
name=gender|type=formfield|subtype=radiobutton|page=1|bottom=50|left=200|width=10|height=10|value=true|buttonid=female|label=female
name=gender|type=formfield|subtype=radiobutton|page=1|bottom=30|left=200|width=10|height=10|value=false|buttonid=male|label=male
```

Parameters

Inline parameter	Value/syntax/description	Remarks
name	The name of the assigned radio button group.	mandatory
type	formfield	mandatory
subtype	radiobutton	mandatory
buttonid	Unique radio button name.	mandatory
page	The page number.	mandatory
bottom, left, width, height	Document coordinates (origin in the lower-left corner), height and width determine the size of the radio button.	mandatory
value	true or false Criterion for selected or not (default is false if omitted).	optional
required	true or false Sets the "required" attribute for the complete radio button group.	optional

Inline parameter	Value/syntax/description	Remarks
readonly	true or false Read only/locked radio button.	optional
tooltip	Sets the tooltip for the radio button.	optional
label	Sets a friendly name (label) for the radio button (is used as tooltip if "tooltip" is not defined).	optional

If the specified radio button group does not exist a new radio button group field will be created in the document. If the buttonid does not exist for the specified radio button group a new radio button (widget) will be added with this value to the group.

The appearance of the radio button cannot be changed via command interface. Only the size of a radio button can be manipulated via width and height parameter.

The commands are processed in the order of the cmd (key) value (in alphabetic order). E.g. cmd_1 will be processed prior to cmd_2, but cmd_10 will be also handled previous to cmd_2 (alphabetic order!). If you select more than one radio button with value=true then the last processed entry is determinative which means that only the last radio button entry (with value=true) will be selected in the document.

Example of document load page

Example HTML page

```
<html>
  <head>
    <title>Simple Document Load Page</title>
  </head>
  <body>
    <p style="border:1px solid gray;background-color:#FFBBBB;">
      <b>REMARK:</b> This page assumes, that the domain www.signdocweb.com is
      accessible...<br>
      <a href="http://www.signdocweb.com">http://www.signdocweb.com</a><br>
      If this is not the case, please edit the source code of this HTML file.
    </p>

    <p style="border:1px solid gray;background-color:lightyellow;">
      <b>Simple Example</b><br>
      Simply open a document by downloading the file and assigning it a DocumentID
      (docid).<br>
      IMPORTANT: It is required, that the document already contains digital signature
      fields to be signed.<br>
      Downloaded file...<br>
      <a href="http://download.srv.softpro.de/testdocs/001.pdf">http://
      download.srv.softpro.de/testdocs/001.pdf<br></a><br>
      - The document ID is set to example_contractid<br>
      <form action="http://www.signdocweb.com/sdweb/load/byurl" target="_blank"
      method="post">
        <input type="hidden" name="docid" value="example_contractid"/>
        <input type="hidden" name="docurl" value="http://download.srv.softpro.de/
        testdocs/001.pdf"/>
        <input type="submit" name="opendoc" value="open document"/>
      </form>
    </p>
  </body>
</html>
```

```

</form>
</p>

<p style="border:1px solid gray;background-color:lightyellow;">
  <b>Example 1</b><br/>
  Opens a document by downloading the file<br/>
  <a href="http://download.srv.softpro.de/testdocs/001.pdf">http://
download.srv.softpro.de/testdocs/001.pdf<br/></a><br/>
  - The document ID is set to example_1<br/>
  - A required signature field named "my_signature" is inserted at the lower left
corner of page 1<br/>
  - The Field A_Personal_ID is filled with the value "123456"<br/>
  - The Field Co_Applicant_Signature is marked as required signature (turns
red)<br/>
  <form action="http://www.signdocweb.com/sdweb/load/byurl" target="_blank"
method="post">
    <input type="hidden" name="docid" value="example_1"/>
    <input type="hidden" name="docurl" value="http://download.srv.softpro.de/
testdocs/001.pdf"/>
    <input type="hidden" name="cmd_1" value="name=my_inserted_signature|page=1|
type=formfield|subtype=signature|bottom=10|left=10|width=150|height=50|required=true"/>
    <input type="hidden" name="cmd_2" value="name=A_Personal_ID|value=123456|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_3" value="name=Co_Applicant_Signature|
value=123456|type=formfield|subtype=signature|required=true"/>
    <input type="submit" name="opendoc" value="open document"/>
  </form>
</p>

<p style="border:1px solid gray;background-color:lightblue;">
  <b>Example 2</b><br/>
  Opens a document by downloading the file<br/>
  <a href="http://download.srv.softpro.de/testdocs/001.pdf">http://
download.srv.softpro.de/testdocs/001.pdf<br/></a><br/>
  - The document ID (docid) is set to "example_2"<br/>
  - The DMS Plugin (dmsid) to use is set to
"de.softpro.sdweb.plugins.impl.FileDms"<br/>
  - A signature field named "my_signature" is inserted at the position of text
"Signature of Applicant" with an x/y offset of 10/-10<br/>
  - The Text Field A_Personal_ID is filled with the value "123456"<br/>
  - The Text Field A_First_Name is filled with the value "John"<br/>
  - The Text Field A_Last_Name is filled with the value "Doe"<br/>
  - The Checkbox Field A_Mandate UpTo5000 is checked<br/>
  - The Metadata Property haircolor is set to the value black<br/>
  <form action="http://www.signdocweb.com/sdweb/load/byurl" target="_blank"
method="post">
    <input type="hidden" name="docid" value="example_2"/>
    <input type="hidden" name="dmsid" value="de.softpro.sdweb.plugins.impl.FileDms"/>
    <input type="hidden" name="docurl" value="http://download.srv.softpro.de/
testdocs/001.pdf"/>
    <input type="hidden" name="cmd_1" value="name=my_inserted_signature|
type=formfield|subtype=signature|searchtext=Signature of Applicant|width=150|height=50|
offsetx=10|offsety=-10|required=true"/>
    <input type="hidden" name="cmd_2" value="name=A_Personal_ID|value=123456|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_3" value="name=A_First_Name|value=John|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_4" value="name=A_Last_Name|value=Doe|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_5" value="name=A_Mandate_UpTo5000|value=true|
type=formfield|subtype=checkbox"/>
    <input type="hidden" name="cmd_6" value="name=haircolor|value=black|
type=metadata"/>
    <input type="submit" name="opendoc" value="open document"/>
  </form>
</p>

```



```

</form>
</p>

<p style="border:1px solid gray;background-color:lightyellow;">
  <b>Example 3</b><br/>
  Opens a document by downloading the file<br/>
  <a href="http://download.srv.softpro.de/testdocs/001.pdf">http://
download.srv.softpro.de/testdocs/001.pdf<br/></a><br/>
  - The document ID is set to example_3<br/>
  - The DMS Plugin (dmsid) to use is set to
  "de.softpro.sdweb.plugins.impl.FileDms"<br/>
  - The Text Field A_Personal_ID is filled with the value "123456"<br/>
  - The Text Field A_First_Name is filled with the value "John"<br/>
  - The Text Field A_Last_Name is filled with the value "Doe"<br/>
  - The Checkbox Field A_Mandate_UpTo5000 is checked<br/>
  - The Text Field C_First_Name is filled with the value "Jane"<br/>
  - The Text Field C_Last_Name is filled with the value "Doe"<br/>
  - The Text Field C_Personal_ID is set to required and maxlength=8<br/>
  - The Checkbox Field C_Mandate_UpTo10000 is checked<br/>
  - The Field Co_Applicant_Signature is set to "Co Applicant's Signature"<br/>
  - The Field Applicant_Signature is set to "Applicant's Signature" and marked as
  required (turns red)<br/>
  - The Metadata Property haircolor is set to the value red<br/>
  <form action="http://www.signdocweb.com/sdweb/load/byurl" target="_blank"
  method="post">
    <input type="hidden" name="docid" value="example_3"/>
    <input type="hidden" name="dmsid" value="de.softpro.sdweb.plugins.impl.FileDms"/>
    <input type="hidden" name="docurl" value="http://download.srv.softpro.de/
testdocs/001.pdf"/>
    <input type="hidden" name="cmd_2" value="name=A_Personal_ID|value=123456|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_3" value="name=A_First_Name|value=John|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_4" value="name=A_Last_Name|value=Doe|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_5" value="name=A_Mandate_UpTo5000|value=true|
type=formfield|subtype=checkbox"/>
    <input type="hidden" name="cmd_6" value="name=haircolor|value=red|type=metadata"/>
  >
    <input type="hidden" name="cmd_7" value="name=Co_Applicant_Signature|
type=formfield|subtype=signature|friendlyname=Co Applicant's Signature"/>
    <input type="hidden" name="cmd_8" value="name=D_Applicant_Signature|
type=formfield|subtype=signature|required=true|label=Applicant's Signature"/>
    <input type="hidden" name="cmd_9" value="name=C_First_Name|value=Jane|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_10" value="name=C_Last_Name|value=Doe|
type=formfield|subtype=textfield"/>
    <input type="hidden" name="cmd_11" value="name=C_Mandate_UpTo10000|value=true|
type=formfield|subtype=checkbox"/>
    <input type="hidden" name="cmd_12" value="name=C_Personal_ID|type=formfield|
subtype=textfield|maxlength=8|required=true"/>
    <input type="submit" name="opendoc" value="open document"/>
  </form>
</p>

</body>
</html>

```

Dynamic tablet screens

Description

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

What can be displayed/hidden?

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

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

```
SDWEB_HOME/tablet_screens/
```

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

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

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

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

Usage

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

- Default layout

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

See SDWEB_HOME/tablet_screens/WacomSTUSeries.-xml

- Custom layout for inserted signature fields

Add the attribute

```
screenlayout=<LayoutID>
```

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


Example

```
name=sig1|page=1|type=formfield|subtype=signature|bottom=10|left=10|
width=150|height=50|screenlayout=piggybank_example
See SDWEB_HOME/tablet_screens/piggybank_example.xml
```

Description of XML elements

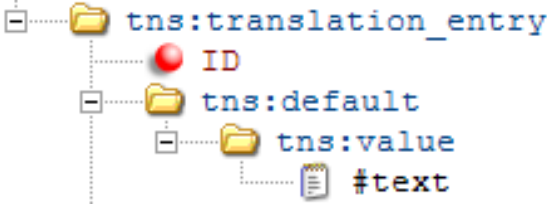
SDWEB_Home/tablet_screens/WacomSTUSeries.xml

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

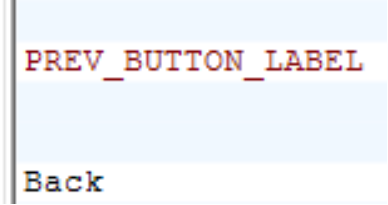
 The language of the button's label can be changed in the TextTranslationTable.xml that is located in the directory sdweb_home/i18n by adding a new value to the translation_entry element.

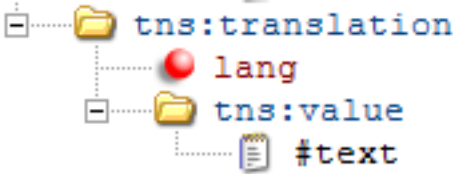
Example

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

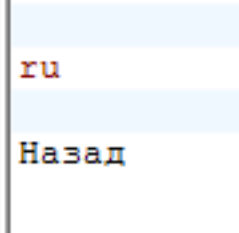


```
tns:translation_entry
├── ID
├── tns:default
│   └── tns:value
│       └── #text
```





```
tns:translation
├── lang
├── tns:value
│   └── #text
```



Description of the attributes

Attribute Description	ActionButton	DefaultFont	Image	TextBox	TextLine	Rectangle
background-color Defines the color of the rectangle's enclosing area.	x		x	x	x	x
blackAndWhiteDithering Defines dithering type for the image when reducing the colors to black and white			x			
border-color Defines the color of the rectangle's border line	x		x	x	x	x
border-width Line width of the border in pixel.	x		x	x	x	x
default-z-order The default z-order of the element. 0 means background.	x		x	x	x	x
font-size Defines dithering type for the image when reducing the colors to black and white						
font-weight either REGULAR or BOLD						
element_id The element ID of a layout element. This ID can be used to reference the element within a SPDynamicContentMap element.	x		x	x	x	x
having_round_corners Defines, if the corners of a rectangle are rounded.	x		x	x	x	x
height The height of an element	x		x	x	x	x
left The left coordinate of an element.	x		x	x	x	x
origin The origin of the elements coordinate system.	x		x	x	x	x
text-color Color of the text.	x			x	x	
text-halign Horizontal alignment of the text.	x			x	x	

Attribute Description	ActionButton	DefaultFont	Image	TextBox	TextLine	Rectangle
text-valign Vertical alignment of the text.				x	x	
top The top coordinate of an element.	x		x	x	x	x
unit Defines the format of coordinates.	x		x	x	x	x
user-z-order The user-defined z-order of the element. 0 means background. The default-order is defined by attribute default-z-order (fixed value) and is used if 2 elements have the same z-order.	x		x	x	x	x
width The width of an element.	x		x	x	x	x
word-wrapping Defines if word-wrapping should be done.				x		

Chapter 3

Remote interface

SignDoc Web RemoteInterface V1.40

The SignDoc Web RemoteInterface allows the communication between the mobile-gui and the hosting application via JavaScript methods/calls.

The hosting application can either be SignDoc Mobile (mobile-app) or a web application that hosts the mobile-gui in an iFrame (web-app).

Same-origin policy

! Because of the 'Same-origin policy' (SOP) in JavaScript the web-app and the mobile-gui (SDWeb) both have to be the same origin to interact with each other.

1. Scenarios in compliance with SOP:
 - web-app and SDWeb are deployed on the same webserver using the same port
 - web-app and SDWeb are deployed on different webserver but are accessed via the same proxy
2. Scenarios in compliance with SOP by reducing the check only to the domain (*):
 - web-app and SDWeb are deployed on different webserver and/or ports in the same domain
3. Scenarios not in compliance with SOP
 - web-app and SDWeb are deployed on webserver in different domains

The check for the SOP compliance can be reduced to the domain by setting the DOM attribute `document.domain` in the mobile-gui AND the hosting web-app HTML page.

Example

SDWeb and web-app are deployed like this:

`http://host1.example.com:8080/sdweb`

`http://host2.example.com:8081/web-app`

In SDWeb the `document.domain` of the mobile-gui can be configured via `sdweb_config.groovy` entry `sdweb.web_page_options.document_domain="example.com"`.

The web-app html page which hosts the mobile-gui inside an iFrame have to set the DOM attribute `document.domain='example.com'`.

Execute actions in mobile-gui

The hosting application can execute actions in the mobile-gui.

Available JavaScript methods are:

```
_spRemote_execute(int actionId, String[] parameters)
(*) _spRemote_execute2(String parameters)
_spRemote_executeBinary(int actionId, String[] parameters, byte[] binaryData)
```

i For the method marked with (*) all parameters have to be url encoded and delimited by |.

Always call these methods inside a try/catch block because the remote interface will throw an exception if there is something wrong with the parameters! .

Before executing actions always check if the action is enabled. Because the action can be temporary disabled when the user interacts with the view you can use the JavaScript setInterval mechanism to wait for execution until the action is enabled again.

Here is an example how to call method.

```
_spRemote_execute(int actionId, String[] parameters)
```

if the mobile-gui is hosted inside an iFrame:

```
function call_spRemote_execute(actionId, params){
var iframe = document.getElementById('sdweb_frame');
try{
    iframe.contentWindow._spRemote_execute(actionId, params);
}
catch(err){
    window.alert(err);
}
}
```

In the following table the actions with the IDs 29, 30, 34, 35, 36, 37, 43, 48, 50, 55, 56, 59, 63, 64 can't be accessed via _spRemote_execute method. They are only used for notifications via method _spRemote_inform(actionId, event, parameters).

Action ID	Action Name	Parameters/Description
1	Open AboutInfo	Action is currently not available
2	Cancel document	Action is deprecated
3	Clear captured field	[0] - (optional) field ID of the capture field which should be cleared. If no parameter is provided the currently selected capture field will be cleared.
5	Finalize document	Action is deprecated
6	Show first page	

Action ID	Action Name	Parameters/Description
7	Highlight fields	[0] - 'true' or 'false'
8	Show last page	
9	Show next page	
12	Show page	Action is currently not available [0] - the page number to show
13	Show previous page	
14	Print document	
15	Scan image	Action is currently not available
17	Zoom in	
18	Zoom by zoomfactor	[0] - zoomfactor A zoomfactor can be an integer value like 25, 50, 75, 100, 125, 150, 175, 200, 225 or one of the string values FIT_TO_WIDTH, FIT_TO_HEIGHT or FIT_TO_SIZE. Note that the corresponding zoomfactor has to be configured in the client configuration: mobile_configuration.xml
19	Zoom out	
20	Open signature fields dialog	
28	Register hosting application	[0] - mobile-app: the mobile device ID or web-app: the browsers user agent [1] - use acknowledge mode ('true' or 'false') Some notifications have to be acknowledged in any case. [2] - version string (in the notation 'V1.8', 'V1.9', ...etc) of the remote interface used in the hosting application. The mobile-gui checks if this parameter convenes to the minimal required version. [3] - mobile-app: the mobile-app environment as JSON string (see Hints and examples , section "mobile-app") or web-app: the web-app environment as JSON string (see Hints and examples , section "web-app")
29	mobile-gui loaded	[0] - version string (in the notation 'V1.8', 'V1.9', ...etc) of the remote interface used in the mobile-gui. [1] - max upload size in bytes (configured in the SignDoc Web server configuration) [2] - cache mobile-gui ('true' or 'false') [3] - capabilities as JSON string (see Hints and examples Capabilities parameter) This notification has to be acknowledged!

Action ID	Action Name	Parameters/Description
30	Capture field clicked	<p>[0] - FIELD_TYPE (0=signature, 1=image, 2=c2s)</p> <p>[1] - FIELD_ID (the internal ID of the capture field)</p> <p>If capture field is of type image these additional parameters are available:</p> <p>[2] - IMG_WIDTH (the width of the image in pixel)</p> <p>[3] - IMG_HEIGHT (the height of the image in pixel)</p> <p>[4] - IMG_FORMAT (the image format: 'jpeg', 'png' or 'tiff')</p>
31	Use mobile-gui toolbar	<p>[0] - 'true' or 'false'</p> <p>(true: use toolbar of mobile-gui; false: hosting application provides a toolbar (native-toolbar))</p>
31	Update capture field	<p>1. For this action the method <code>_spRemote_executeBinary(int actionId, String[] parameters, byte[] binaryData)</code> has to be used.</p> <p>Parameters:</p> <p>[0] - FIELD_ID (the internal ID of the field)</p> <p>[1] - CAPTURE_ACTION (0=Update or 1=Cancel)</p> <p>If CAPTURE_ACTION = 0:</p> <p>[2] - FORMAT ('a'(simple) or 'B'(extended - header and timestamps) for a signature or 'image' for an image)</p> <p>binaryData: The binary data of the signature/image</p> <p>2. For backward compatibility and Windows 8 support the methods <code>_spRemote_execute(int actionId, String[] parameters)</code> and <code>_spRemote_execute2(String parameters)</code> are also available.</p> <p>But the maximal supported length of base64 encoded signature/image data is 65536 bytes (depending on the browser)!</p> <p>[0] - FIELD_ID (the internal ID of the field)</p> <p>[1] - CAPTURE_ACTION (0=Update or 1=Cancel)</p> <p>[2] - FORMAT (optional) ('a'(simple) or 'B'(extended - header and timestamps) for a signature or 'image' for an image. When using signature format 'a' this parameter can be omitted)</p> <p>[3] - DATA (Base64 encoded signature/image data or an empty string for Cancel action.</p> <p>Encoded signature data must begin with the prefix character 'a' or 'B' to identify the signature format.)</p> <p>Note This action can only be called after the event 30 (Capture field clicked) is send.</p>

Action ID	Action Name	Parameters/Description
33	Display dialog in mobile-gui	<p>[0] - ID (ID of the dialog, can be used for identification when action 34 sends notification)</p> <p>[1] - TYPE (0=Info, 1=Warning, 2=Error or 3=Custom -> types 0, 1 and 2 are simple message dialogs with a OK button)</p> <p>[2] - MESSAGE (the message to be displayed)</p> <p>[3] - CUSTOM_BUTTONS (if TYPE = 3: comma separated list of buttons: 0=Yes, 1=No, 2=OK, 3=Cancel, 4=Close, 5=Accept, 6=Reject or an empty string)</p> <p>[4] - CUSTOM_TITLE (if TYPE = 3: custom title string or an empty string)</p>
34	Dialog button clicked	<p>[0] - ID (ID of the corresponding dialog - see action 33)</p> <p>[1] - BUTTON (ID of the clicked button - 0=Yes, 1=No, 2=OK, 3=Cancel, 4=Close, 5=Accept, 6=Reject)</p> <p>Notes:</p> <p>Notification for button clicks of dialogs triggered via action 33.</p> <p>If hosting application is a web-app this notification is also send for button clicks in capture dialogs:</p> <p>dialogId: SignatureCapture</p> <p>Buttons:</p> <p>2 = OK (Capture dialog was closed by clicking OK button)</p> <p>3 = Cancel (Capture dialog was closed by clicking CANCEL button)</p> <p>dialogId: ImageCapture</p> <p>Buttons:</p> <p>2 = OK (Capture dialog was closed by clicking OK button)</p> <p>3 = Cancel (Capture dialog was closed by clicking CANCEL button)</p> <p>dialogId: QuestionableMatch (QuestionableMatch dialog is only displayed when a custom SignatureArchivePlugin is used that returns a corresponding return code)</p> <p>Buttons:</p> <p>5 = Accept (Dialog was closed by clicking Accept button)</p> <p>6 = Reject (Dialog was closed by clicking Reject button)</p> <p>Important: If a SignatureArchivePlugin is used the notification of button 2=OK from dialog SignatureCapture is no guarantee that the signature field is signed!</p>
35	Localized data loaded	<p>[0] - LOCALE</p> <p>Each of the following localized data is embedded in " characters.</p> <p>[1] - CONFIRM_CANCEL_DOCUMENT_MSG</p> <p>[2] - CONFIRM_CANCEL_DOCUMENT_TITLE</p> <p>[3] - CONFIRM_FINALIZE_DOCUMENT_MSG</p> <p>[4] - CONFIRM_FINALIZE_DOCUMENT_TITLE</p> <p>[5] - CONFIRM_CLEAR_SIGNATURE_MSG</p> <p>[6] - CONFIRM_CLEAR_SIGNATURE_TITLE</p> <p>[7] - CONFIRM_CLEAR_IMAGE_MSG</p> <p>[8] - CONFIRM_CLEAR_IMAGE_TITLE</p> <p>[...] - can be extended in the future</p>

Action ID	Action Name	Parameters/Description
36	Document loaded	[0] - number of document pages [1] - document ID [2] - context URL [3] - session ID [4] - download URL
37	Page changed	[0] - the current page number
38	Add capture field	Currently only supported for mobile-app
39	Download document	Action has to be implemented by the hosting application
40	Open document	Action has to be implemented by the hosting application
41	E-Mail	Action has to be implemented by the hosting application
42	Show help	
43	Enable native-toolbar	This action is only used for sending enabled state notifications in order to inform the hosting application if the native-toolbar has to be enabled or disabled.
44	Show SoftKeyboard	[0] - visibility of SoftKeyboard ('true': visible; 'false': hidden) [1] - the initiator of the action call ('mobilegui' or 'nativeapp') This action is used by both the mobile-gui (to force the SoftKeyboard to popup) and the mobile-app (to inform the mobile-gui when the SoftKeyboard visibility state changed).
45	Screen orientation changed	Action is deprecated [0] - the new screen orientation ('portrait' or 'landscape') [1] - the new visible width (in pixel) of the web view [2] - the new visible height (in pixel) of the web view
47	Update image external	This action can be used to set an image to an unsigned image field or to clear the image of a signed image field via the remote interface (without user action in the mobile-gui). For this action the method _spRemote_executeBinary(int actionId, String[] parameters, byte[] binaryData) has to be used. [0] - FIELD_ID (the internal ID of the field) [1] - CAPTURE_ACTION (0=Insert or 1=Clear) If CAPTURE_ACTION = 0: binaryData: The binary data of the image Note It is recommended to use this action after the event 36 (Document loaded) is received.
48	Refresh WebView	This action is used to inform the mobile-app to refresh the WebView component (this is needed on android devices to get rid of additional content space when an image is zoomed out).

Action ID	Action Name	Parameters/Description
49	Request document snapshot	<p>Request a snapshot of the current document which can be accessed via sdweb webservice after action 50 is received.</p> <p>[0] - the snapshot action (0=download, 1=email, 2=print)</p> <p>[1] - options (optional) as JSON string (for example to get a specific version of the document)</p> <p>Note This action can only be used if the current document is available in the mobile-gui (the event 36 (Document loaded) is received).</p>
50	Document snapshot available	<p>Notification for the mobile-app that the requested snapshot is available on the sdweb server.</p> <p>[0] - the reference ID of the snapshot (used for the webservice call)</p>
51	Page info	Action is currently not available
52	Update capture field (EXT)	<p>1. For this action the method <code>_spRemote_executeBinary(int actionId, String[] parameters, byte[] binaryData)</code> has to be used.</p> <p>Parameters:</p> <p>[0] - FIELD_ID (the internal ID of the field)</p> <p>[1] - CAPTURE_ACTION (0=Update or 1=Cancel)</p> <p>If CAPTURE_ACTION = 0:</p> <p>[2] - FORMAT 'a'(simple) or 'B'(extended - header and timestamps) for a signature, 'image' for an image or 'c2s' for a c2s signature</p> <p>[3] - OPTIONS options as JSON string - see Hints and examples section "Options parameter")</p> <p>binaryData: The binary data of the signature/image (can be empty for a c2s signature)</p> <p>2. For backward compatibility and Windows 8 support the methods <code>_spRemote_execute(int actionId, String[] parameters)</code> and <code>_spRemote_execute2(String parameters)</code> are also available.</p> <p>But the maximal supported length of base64 encoded signature/ image data is 65536 bytes (depending on the browser)!</p> <p>[0] - FIELD_ID (the internal ID of the field)</p> <p>[1] - CAPTURE_ACTION (0=Update or 1=Cancel)</p> <p>[2] - FORMAT 'a'(simple) or 'B'(extended - header and timestamps) for a signature, 'image' for an image or 'c2s' for a c2s signature.</p> <p>[3] - DATA Base64 encoded signature/image data or an empty string for Cancel action.</p> <p>Encoded signature data must begin with the prefix character 'a' or 'B' to identify the signature format (can be empty for a c2s signature).</p> <p>[4] - OPTIONS (options as JSON string - see Hints and examples, section "Options parameter")</p> <p>Note: This action can only be called after the event 30 (Capture field clicked) is send.</p> <p>Action is available since V1.22.</p>

Action ID	Action Name	Parameters/Description
53	Zoom to width	Action is available since V1.23.
54	Validate and finalize	<p>[0] - options (optional) as JSON String in the format {"ShowConfirmation":"True"}</p> <p>ShowConfirmation - specify if the mobile-gui should display a confirmation dialog before the document is finalized.</p> <p>Possible values are True or False. Note that there is a configuration entry for displaying the finalize confirmation dialog which will affect the display of the dialog additionally.</p> <p>Note This action replaces action 5 (Finalize document). Before the document is finalized it is validated.</p> <p>Action is available since V1.24.</p>
55	InputField displayed	<p>Notification for the hosting application that a text field is displayed in 'Popup' mode for entering text.</p> <p>If the text field is displayed in 'Dialog' mode this notification is not send!</p> <p>Currently this notification is only used by the Windows mobile-app to avoid that the popup is hidden by the soft-keyboard of the mobile device.</p> <p>[0] - Parameters as JSON string in the format {"Id":"XYZ", "Type":"Text", "Width":200, "Height":80, "Left":40, "Top":550}</p> <p>Id - the field ID</p> <p>Type - the field type (currently only 'Text' is supported)</p> <p>Width - the width of the field in pixel</p> <p>Height - the height of the field in pixel</p> <p>Left - the left position of the field</p> <p>Top - the top position of the field</p> <p>Action is available since V1.26</p>
56	Application exits	<p>Notification for the hosting application that the mobile-gui exits by calling another url in the web view.</p> <p>[0] - the exit url which is called by the mobile-gui</p> <p>Action is available since V1.27</p> <p>Note This notification has to be acknowledged!</p>
57	Cancel document	<p>[0] - options (optional) as JSON String in the format {"ShowConfirmation":"True"}</p> <p>ShowConfirmation -specify if the mobile-gui should display a confirmation dialog before the document is canceled.</p> <p>Possible values are True or False.</p> <p>Note This action replaces action 2 (Cancel document).</p> <p>Action is available since V1.28</p>

Action ID	Action Name	Parameters/Description
58	Process capture field	<p>This action allows the interaction with a capture field in the document. Depending on the corresponding state of the capture field and the specified FieldAction different field actions can be triggered. See more information below.</p> <p>[0] - options as JSON String in the format {"FieldId":"xyz", "FieldAction":"Capture", "ShowClearConfirmation":"True"}</p> <p>FieldId - the field ID of the capture field</p> <p>FieldAction - the action which should be executed on the capture field.</p> <p>Possible values are:</p> <p>Capture: starts capture process if field is not signed</p> <p>Clear: clears capture field if it is signed</p> <p>Clear-Capture: clears capture field if it is signed and/or starts capture process</p> <p>ShowClearConfirmation - specify if the mobile-gui should display a confirmation dialog before the capture field is cleared.</p> <p>Possible values are True or False.</p> <p>Action is available since V1.28.</p>
59	Initial UI ready	<p>Notification for the hosting application that the initial UI processing has finished and the first page/image of the document is displayed.</p> <p>Action is available since V1.29.</p>
60	Cancel capturing	<p>Action that cancels the current capture process.</p> <p>Action is available since V1.30.</p>
61	Scroll document view	<p>Scrolls the document view by specified amount of pixel in specified orientation.</p> <p>[0] - scroll options as JSON String in the format {"Orientation":"Vertical", "Pixel":100}</p> <p>Orientation - Vertical or Horizontal</p> <p>Pixel - Numeric amount of pixel (use negative amount for scrolling up or scrolling left)</p> <p>Note In IE7 and IE8 dialogs which overlay the document view are also scrolled.</p> <p>Action is available since V1.31.</p>
62	Show audit trail	<p>Displays the audit trail in a new browser window/tab.</p>

Action ID	Action Name	Parameters/Description
63	Field updated	<p>Notification for the hosting application that the user has updated a field.</p> <p>[0] - field information as json string in the format <code>{"FieldId":"xyz", "FieldType":"Signature", "FieldAction":"Captured"}</code></p> <p>FieldId - the field ID of the updated field (in case of a radiobutton the field ID is the name of the radiobutton group followed by the index of the radiobutton in brackets, for example Group[1]).</p> <p>FieldType - the type of the updated field. Possible values are: Signature: a signature field Image: an image field C2S: a Click-to-Sign field Text: a text field Checkbox: a checkbox field Radiobutton: a radiobutton field</p> <p>FieldAction - the update action of the field. Possible values are: Captured: the field was captured (signature, image and c2s) Cleared: the field was cleared (signature, image and c2s) Updated: the text field was updated (text) Selected: the field was selected (checkbox and radiobutton) Unselected: the field was unselected (checkbox)</p> <p>FieldValue - the value of the field (currently only set for FieldType Text)</p> <p>Action is available since V1.36.</p>
64	General notification	<p>Notification for the hosting application that something happened.</p> <p>[0] - notification information as json string in the format <code>{"NotificationID":"xyz", "Parameters":{...}}</code></p> <p>See Hints and examples, section "Options parameter" for a list of all notifications and their parameters.</p>
65	Select field	<p>Selects corresponding field and scrolls it into view</p> <p>[0] - the field ID</p>

Notification about action events

The hosting application is informed about events by the mobile-gui.

For sending notifications there are different approaches available for web-app and mobile-app.

mobile-app

- 'Windows 8' mobile-app:

The notification is sent by calling the 'Windows 8' specific JavaScript method 'window.external.notify(urlString)'.

The structure of the urlString parameter is:


sdweb://localhost/action?actionid=xx&event=yy¶ms=zzz;aaa;...etc

- All other mobile-apps:

The notification is sent via an URL change.

The structure of an URL is:

sdweb://localhost/action?actionid=xx&event=yy¶ms=zzz;aaa;...etc

 All URL calls using protocol sdweb have to be blocked by the mobile-app, no matter if the event is consumed or not!

web-app

The notification is sent by calling the JavaScript method

```
_spRemote_notification(int actionId, int eventId, String[] parameters)
```

which has to be provided globally in the HTML <Head> section by the hosting html page of the web-app.

Possible events and their parameters

Event ID	Event Name	Parameters/Description
1	EXECUTED	Parameters are depending on the action. See explanation below the table
2	ENABLED_STATE_CHANGED	[0] - 'true' or 'false' (means: enabled=true or enabled=false)
3	VISIBLE_STATE_CHANGED	[0] - 'true' or 'false' (means: visible=true or visible=false)
4	LABEL_CHANGED	[0] - the label currently not available
5	TOOLTIP_CHANGED	[0] - the tooltip currently not available
6	TOGGLE_STATE_CHANGED	[0] - 'true' or 'false' (means: active=true or active=false) currently not available

Currently the event with ID 1 (see table above) will only be sent by following actions:

29 - mobile-gui loaded

30 - Capture field clicked

34 - Dialog button clicked (notification only for dialogs requested via action 33)

35 - Localized data loaded


36 - Document loaded

- 37 - Page changed
- 44 - Show SoftKeyboard
- 48 - Refresh WebView
- 50 - Document snapshot available
- 55 - InputField clicked
- 56 - Application exits
- 59 - Initial UI ready
- 63 - Field updated
- 64 - General notification

Acknowledge notifications

Because the url change listener from the mobile-app may not receive events if they are sent in very short intervals an acknowledge mechanism is implemented.

If acknowledge mode is enabled each event/notification sent from the mobile-gui has to be acknowledged first by the hosting application before the next event will be sent. The acknowledge mode can be enabled by setting the corresponding parameter in action 28.


 The notification with actionId 29 (mobile-gui loaded) and actionId 56 (Application exits) have to be acknowledged by the hosting application in any case.

In order not to block the mobile-gui completely when an event is not acknowledged by the hosting application, a timeout mechanism is implemented after which the next event is sent automatically.

If an event is not acknowledged inside the corresponding timeframe (1500ms) it will be re-send up to a configurable max number of retries. Per default it is configured to be re-sent once.

The acknowledge is done by calling one of the following JavaScript methods:

```
spRemote_acknowledge(int actionId, int event)
spRemote_acknowledge2(String parameter)
```

 For the method containing the parameters 'actionId' and 'event' these have to be delimited by | (i.e. 29|1).

mobile-app document sequence example

1. SDWeb is called from post request (WebPortal in mobile-app).
2. Mobile-gui is loaded in WebView of mobile-app.
3. Notification from mobile-gui:

```
sdweb://localhost/action?actionid=29&event=1&params=...
```

4. Mobile-app registers itself by calling

```
_spRemote_execute(28, [DEVICE_ID, ACKNOWLEDGE_MODE, VERSION])
```

5. If mobile-app provides its own toolbar the toolbar of the mobile-gui can be disabled by calling

```
_spRemote_execute(31, [false])
```

6. The user wants to capture a signature (he clicks in a signature field).**7. Notification from mobile-gui:**

```
sdweb://localhost/action?actionid=30&event=1&params=0;FIELD_ID
```

8. After capturing a signature mobile-app calls

```
_spRemote_execute(32, [FIELD_ID, '0', 'a', SIGNATURE_DATA])
```

or when the capturing is canceled SignDocMobile calls

```
_spRemote_execute(32, [FIELD_ID, '1', ''])
```

Hints and examples

JavaScript example of executing action 28 - Register hosting application

```
var params=[];
params[0]='deviceid123';
params[1]='true';
params[2]='v1.28';
params[3]= ...app environment...
try{
    _spRemote_execute(28,params);
}
catch(err){
    window.alert(err);
}
```

JavaScript example of executing action 33 - Display dialog in mobile-gui

```
// 1. display info message
var params=[];
params[0]='dialogId-123';
params[1]='0';
params[2]='Test info message';
params[3]='';
params[4]='';
try{
    $wnd._spRemote_execute(33, params);
}
catch(err){
    window.alert(err);
}

// 2. display cancel dialog
// cancelMsg and cancelTitle needs to be read when mobile-gui sends LocalizedDataLoaded
// notification:
// sdweb://localhost/action?actionid=35&event=1&params=en,'the cancel message','the
// cancel title',... etc
var cancelMsg;
var cancelTitle;

var params=[];
```

```

params[0]='dialogId-456';
params[1]='3';
params[2]=cancelMsg;
params[3]='0,1';
params[4]=cancelTitle;
try{
    $wnd._spRemote_execute(33, params);
}
catch(err){
    window.alert(err);
}
// 2.1 receive Dialog button clicked notification from Cancel dialog

// User clicked Yes button:
// sdweb://localhost/action?actionid=34&event=1&params=dialogId-456,0

// execute action 2 - 'Cancel document'
try{
    var params=[];
    $wnd._spRemote_execute(2, params);
}
catch(err){
    window.alert(err);
}

// User clicked No button:
// sdweb://localhost/action?actionid=34&event=1&params=dialogId-456,1

// do nothing

```

App environment

- mobile-app

Example of a valid JSON string format:

```

{"AppVersion":"1.2.0", "ScreenSize":"1024x768@2", "OS":"iPhone OS",
 "AppID":"de.softpro.SignDocMobile", "DeviceModel":"iPhone4,1",
 "CPUModel":"12.6", "CPUClock":"530000000", "DeviceVendor":"Apple", "CaptureType":
 ["SPen", "CPen", "Mouse"], "ConfigLinkID":"000"}

```

- web-app

Example of a valid JSON string format:

```

{"WebApp":{"SignatureCaptureDevice":"Plugin",
 "SignatureCaptureConfig":"SD_SignatureCaptureDialog",
 "ImageCaptureDevice":"Plugin", "ImageCaptureConfig":"SD_ImageCaptureDialog"}}

```

Available parameters and their values:

Parameter	Value
SignatureCaptureDevice	Mandatory parameter Plugin External HTML5 Auto
SignatureCaptureConfig	Name of the corresponding configuration element in mobile_configuration.xml that is used by the signature capture dialog Default for Plugin: SD_SignatureCaptureDialog for Plugin Default for HTML5: SD_SignatureCaptureDialogHtml5 for HTML5
ImageCaptureDevice	Mandatory parameter Plugin External HTML5 Auto

Parameter	Value
ImageCaptureConfig	Name of the corresponding configuration element in mobile_configuration.xml that is used by the image capture dialog Default for Plugin: SD_ImageCaptureDialog) For HTML5 there is no custom configuration possible.
C2SCaptureDevice	Plugin External Default: Plugin
ResultUrl	True False Default: True

Description of parameters and values:

Parameter	Description
Plugin	Data is captured with Kofax SignDoc Device Support . If it is not available capturing is not possible and an error is shown.
External	Data is captured/provided by the web-app
HTML5	Signature or image is captured via HTML5 canvas
Auto	Signature or image is captured with Kofax SignDoc Device Support , if it is not available signature or image is captured with HTML5. Note that once the capturing falls back to HTML5 no check for available capture mechanism is done anymore until the page is reloaded.
ResultUrl	Triggers if the result-url should be displayed after exiting the application via Finalize, Cancel, Session Timeout, etc

Options parameter for action 52 (Update capture field)

Example of a valid JSON string format:

```
{"Signer":"Hugo Habicht", "SignTime":"24.12.2012 10:30:23"}
```

Available parameters and their values:

Parameter	Value
Signer	The signer name (only available for mobile-app)
SignTime	The signing time (only available for mobile-app)
Payload	Additional signing information for later use (i.e. in a custom plugin)

Capabilities parameter for action 29 (mobile-gui Loaded)

Example

```
{
  "capabilities": {
    "actions": [
      28, 29, 1, 2, 3, 5, 6, 7, 8, 9, 13, 17, 18, 19, 20, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
      47, 48, 49, 50, 51, 52, 53, 54
    ]
  }
}
```

actions: all action IDs supported in the current RemoteInterface version of the mobile-gui

All notifications and their parameters for action 64

Document validation

Document is validated. This is currently done before document is sent to archive/DMS. If the validation fails the document is not sent to the archive/DMS.

NotificationId:

DocumentValidation

Parameters:

"State": "Success" or "Failure"

"Message": "..." // the failure message or empty if success

Document archiving/DMS status

Document is sent to archive/DMS.

NotificationId:

DocumentArchiving

Parameters:

"State": "Success" or "Failure"

Message": "..." // the failure message or empty if success

Field selected

An interactive field is selected.

NotificationId:

FieldSelected

Parameters:

"State": "True" or "False" // field selected or unselected

"FieldId": "..." // the selected or unselected field ID/name

"HtmlId": "..." // the html ID of the selected field/div

Field clicked

An interactive field is clicked by the user.

NotificationId:

FieldClicked

Parameters:

"FieldId": "..." // the clicked field ID/name

"HtmlId": "..." // the html ID of the clicked field/div

Field canceled

The editing of an interactive field is canceled by the user.

NotificationId:

FieldCanceled

Parameters:

:"FieldId": "..." // the canceled field ID/name

:"HtmlId": "..." // the html ID of the canceled field/div

Session expired

The current session has expired and the application will either exit immediately or (if configured) after user clicks OK button in the session expired info dialog.

NotificationId: SessionExpired

Server down

The server can't be connected anymore (server down or connection/network issue) and the application will exit immediately.

NotificationId: ServerDown

Version history

Version	Description
V1.0	Initial version
V1.1	Added chapter "Hints and examples" Added try/catch hint New remote actions available for SignDocMobile: 'Download document' and 'Zoom by zoomfactor'
V1.2	Added new action 32 (Update capture field) Added new parameters to action 30 (Capture field clicked) for future use
V1.3	Added new action 33 (DisplayGWTDIALOG) Added new action 34 (DialogButtonClicked) Added new action 35 (LocalizedDataLoaded) Added new Hints in chapter "Hints and examples" Added info that signature data has to begin with character 'a'
V1.4	Added new method _spRemote_executeBinary(int actionId, String[] parameters, byte[] binaryData)

Version	Description
V1.5	Refactored action 32 (Update capture field)
V1.6	Added new action 36 (DocumentLoaded) Added new action 37 (PageChanged)
V1.7	Added acknowledge mode parameter to action 28 RegisterSignDocMobile Added part "Acknowledge notifications"
V1.8	Added version string parameter to action 28 RegisterSignDocMobile
V1.9	Action 7 (Highlight fields) is available now New params for action 36 (DocumentLoaded) Changed state of action 31 (Use GWT-Client toolbar) to not supported because of unresolved issues regarding the native toolbar.
V1.10	Updated acknowledge mechanism: If an event is not acknowledged inside the corresponding timeframe (1500ms) it will be resend up to a configurable max number of retries. Per default it is configured to be resend once.
V1.11	Added new action 38 (AddCaptureField) Added new action 39 (DownloadDocument) Added new action 40 (OpenDocument) Added new action 41 (E-Mail) Updated action 29 (GWT-Client loaded) parameter Removed actions from list which are not accessible by SignDoc Mobile
V1.12	Added new action 42 (ShowHelp)
V1.13	Added new action 43 (NativeToolbar) Added new action 44 (SoftKeyboard) Added new action 45 (ScreenOrientationChanged)
V1.15	SDWeb server supports extended signature 'B' format Updated comments for action 32
V1.16	Added new action 47 (UpdateImageFieldExternal) Added note in comments for action 32
V1.17	Action 45 (ScreenOrientationChanged) is deprecated and will not be used anymore by the GWT-Client. The GWT-Client recognizes a screen orientation change by itself now. Extended parameter of action 18 (Zoom by zoomfactor)
V1.18	Added support for Windows 8 Apps: _spRemote_execute2(String parameters), _spRemote_acknowledge2(String parameter)
V1.19	New param for action 32 (2. backward compatibility)
V1.20	Added new action 48 (RefreshWebView) New param for 28 (RegisterSignDocMobile) containing the app environment New chapter "App environment (JSON string format)"

Version	Description
V1.21	Added new action 49 (RequestDocumentSnapshot) Added new action 50 (DocumentSnapshotAvailable)
V1.22	Added new action 52 (update capture field (EXT))
V1.23	Deprecated action 5 (Finalize document) Added new action 54 (Validate and finalize) which has to be used instead of action 5 when RemoteInterface version >= V1.24
V1.24	Deprecated action 5 (Finalize document) Added new action 54 (Validate and finalize) which has to be used instead of action 5 when RemoteInterface version >= V1.24
V1.25	Added new parameter for action 29 ((GWT-Client(Module) loaded))
V1.26	Added new action 55 (InputField clicked)
V1.27	Added new action 56 (Application exits)
V1.28	Desktop browser support (introduced module names: hosting application, web-app, mobile-app, native-toolbar and mobile-gui) Added new chapter "Dictionary" Deprecated action 2 (Cancel document) Added new action 57 (Cancel document) which has to be used instead of action 2 when RemoteInterface version >= V1.28 Added new action 58 (Process capture field) Updated parameter of action 3 (Clear captured field)
V1.29	Added new action 59 (Initial UI ready) Updated hints in chapter "Execute actions in mobile-gui"
V1.30	Added new action 60 (Cancel capturing) Updated notes for action 34 (Dialog button clicked)
V1.31	Added new action 61 (Scroll document view)
V1.32	Added new parameter for action 28 (RegisterSignDocMobile)
V1.33	Added HTML5 as possible value for SignatureCaptureDevice parameter of action 28 (RegisterSignDocMobile)
V1.34	New field type 2=c2s-field in action 30 (Capture field clicked) Added payload option for action 52 (update capture field) - see Hints and examples section "Options parameter" New format 'c2s' option in action 52 (update capture field (EXT)) Added new parameter C2SCaptureDevice in Hints and examples , section "web-app" Changed parameter name NATIVE to EXTERNAL in chapter "Hints and examples", web-app Added new parameter C2SCaptureDevice in Hints and examples , section "web-app"
V1.35	Added new action 62 (Show audit trail)
V1.36	Added new action 63 (Field updated)

Version	Description
V1.37	Added new parameter value 'Auto' in chapter "Hints and examples", web-app Added new parameter C2SCaptureDevice in Hints and examples , section "web-app" Added new parameter C2SCaptureDevice in Hints and examples , section "web-app" Added new parameter 'ResultUrl' in chapter "Hints and examples", web-app Added new parameter C2SCaptureDevice in Hints and examples , section "web-app" Action 14 (print document) is available now Updated comment for action 55 (InputField displayed)
V1.38	Added infos about the 'Same Origin Policy'. New action 64 (General notification). Added options parameter for action 64. New action 65 (Select field). New json parameter FieldValue for action 63.
V1.39	Updated infos about the 'Same Origin Policy'. Added options parameter FieldCanceled for action 64.
V1.40	Added options parameter 'SessionExpired' and 'ServerDown' for action 64. Added new parameter values 'Auto' and 'HTML5' for image capturing in chapter "Hints and examples", web-app Added new parameter C2SCaptureDevice in Hints and examples , section "web-app"

Dictionary

mobile-gui

The web client user interface of SignDoc Web which provides the RemoteInterface and is specially designed for the need of mobile devices.

mobile-app

The native app on a mobile device which hosts the mobile-gui in a WebView component. It is also called SignDoc Mobile.

web-app

The web application which hosts the mobile-gui in an iFrame.

hosting application

The application which hosts the mobile-gui. This can be either a mobile-app or a web-app.

web-toolbar

The toolbar of the mobile-gui.

native-toolbar

The toolbar of the hosting application which replaces the web-toolbar of the mobile-gui.

Chapter 4

REST interface

The SignDoc Web REST Interface provides a simplified possibility to work with PDF and (restricted for) TIFF documents from different clients via HTTP protocol.

According to the REST architecture, a RESTful web service should not keep a client state on the server. This restriction is called Statelessness. This has the advantage, that web services can treat each method request independently and there is no need to maintain the client's previous interactions.

Sometimes it makes sense to break this restriction. SignDoc Web provides a service which allows to upload one or more documents for further processing. The uploaded documents are not stored permanently in SignDoc Web usually, but rather only temporarily for processing by the client. This is the reason why it is not very helpful to handle documents stateless in the SignDoc Web REST service.

Usually a document is uploaded and maybe prepared in the first call. The uploaded document is stored in a server session. The next document related calls are processed with the document which is stored in the client specific session workspace. After processing the document is removed from the session workspace again.

The client must pass its context to the REST interface. In this case the context is the document ID and the session identifier. The document ID is either predefined or created by the server and returned in the response of the document upload request. The session ID is returned in the JSESSIONID cookie of the response.

Example

1. The document is uploaded to SignDoc Web for processing:

```
POST http://localhost:6610/sdweb/rest/v5/documents?init=true
```

(body contains the document, content type has multipart/form-data)

The response contains the document ID:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><restLoadId
xmlns="http://www.kofax.com/ksd/sdweb/rest/v5"><type>DOCID</
type><value>83501</value></restLoadId>
```

The response header contains the session ID:

```
Set-Cookie=[JSESSIONID=5B29777C5A65E50BE24B3EA715454135;path=/
sdweb;HttpOnly]
```

2. The document is updated.

Update the value of a text field:

```
PUT http://localhost:6610/sdweb/rest/v5/documents/83501/textfields/text-1
```

The body contains the value to be changed:

```
{"restTextFieldInput":{"value":"abc123"}}
```

The request header contains the session ID:

Cookie: JSESSIONID=5B29777C5A65E50BE24B3EA715454135

3. The document is removed from the session:

DELETE http://localhost:6610/sdweb/rest/v5/documents/83501

The request header contains the session ID:

Cookie: JSESSIONID=5B29777C5A65E50BE24B3EA715454135

A document can be loaded into workspace from different sources. The document can be uploaded directly to the server (e.g. via REST API) or can be loaded from a specific URL. In addition it is possible to load a raw PDF document which was deposited in SignDoc Web before as a so-called document template.

A preloaded document can be edited directly after upload or it can be handled delayed without having the uploaded document in the session workspace of the requesting client. For delayed processing the uploaded document is only available then in a temporary storage.

In order to make it available for editing in the session workspace for a specific client he must activate the document (load into session workspace) by an additional initialization call via reference ID which was (optionally) returned by the preload call (init=false). The client can request any document information or he can modify the document. The document can be edited by a client until he removes the document explicitly from the session workspace or until session timeout occurs.

The information exchanged between client and application server is typically in JSON format, but could be also XML.

SignDoc Web Server RESTful web services are realized with the JAX-RS Reference Implementation "Jersey" from Oracle. Jersey allows to use various JSON notations. Each of these notations serializes JSON in a different way. SignDoc Web accepts and produces the MAPPED JSON notation (see [JSON Notations](#)) with two additional characteristics.

1. The root element (normally declared with @XmlRootElement) is included in the JSON string.
2. Single list or array elements are also surrounded with square brackets '[' and '']

The notation will be described using a simple example. Following are JAXB beans, which will be used.

Simple address bean

```
@XmlRootElement
public class Address {
    public String street;
    public String town;
    public Address() {}
    public Address(String street, String town) {
        this.street = street;
        this.town = town;
    }
}
```

Contact bean

```
@XmlRootElement
public class Contact {
```

```
public int id;
public String name;
public List<Address> addresses;
public Contact() {};
public Contact(int id, String name, List<Address> addresses) {
    this.name = name;
    this.id = id;
    this.addresses = (addresses != null) ? new LinkedList<Address>(addresses) :
null;
}
}
```

The following text will be mainly working with a contact bean initialized with:

Initialization

I.e. contact bean with


id=2, name="Bob"

containing a single address

(street="Long Street 1",town="Short Village")

JSON expression produced using mapped notation

```
{"contact": {"id": "2",
"name": "Bob",
"addresses": [{"street": "Long Street 1",
"town": "Short Village"}]}}
```

 "contact" as root element is included as well as the square brackets [...] around the addresses element.

The client access to addresses elements is then equivalent if another address was added. With

```
contact.addresses.add(new Address("Short Street 1000", "Long Village"));
```

you would get

```
{"contact": {"id": "2",
"name": "Bob",
"addresses": [{"street": "Long Street 1", "town": "Short Village"},
{"street": "Short Street 1000", "town": "Long Village"}]}}
```

If you access

"Short Village"

value e.g. from a JavaScript client, you will write

```
addresses[0].town
```

Empty lists or arrays are not written to result.



```
{
  "contact": {
    "id": "2",
    "description": "Lorem ipsum dolor sit amet, \n consectetur adipiscing elit..."
  }
}
```

```
{
  "contact": {
    "id": "2",
    "description": "Lorem ipsum dolor sit amet, \n consectetur adipiscing elit..."
  }
}
```

If text with the line breaks is needed to be inserted in HTML, it should be prepared properly.

REST URL

RESTful web services can be accessed via a REST path which is appended to the SignDoc Web context. It is followed by the SignDoc Web REST version number and the requested resource with any necessary parameters.

The URL to the SignDoc Web API has the following syntax:

`scheme://domain:port/path?query_string`

whereas `path` is divided in the parts

`context/rest/version-number/resource`

Part	Description
context	The context is <code>sdweb</code> by default.
rest	The rest part is fixed.
version-number	The version-number is <code>v5</code> .
resource	The <code>resource</code> part identifies the requested resource, for example documents.

Example

`http://localhost:6610/sdweb/rest/v5/documents`

REST error response

HTTP is based on the exchange of representations, and that applies to errors as well.

When a server encounters an error, either because of problems with the request that a client submitted or because of problems within the server, always return a representation that reflects the state of the error condition. This includes the response status code, response headers, and a body containing the description of the error.

For errors due to client inputs, return a representation with a 4xx status code. For errors due to server implementation or its current state, return a representation with a 5xx status code.

In both cases, a Date header with a value indicating the date-time is included at which the error occurred. The date is returned in RFC 1123 format (see [Date/Time formats](#)).

In case of an error the response body contains a RestMessages element.

In case of an error the response body contains a RestMessage element.

RestMessageList

- **list** (RestMessage, optional): List of RestMessage elements

RestMessage

- **code** (number): SignDoc Rest message code
- **message** (string): Message description
- **type** (RestMessage.TYPE): Message type. Possible values: ERROR, WARNING, INFO

Example response body (JSON)

```
{
  "restMessageList" : {
    "list" : [ {
      "code" : 105,
      "message" : "Requested page number range string '1-5' does not contain valid
numbers",
      "type" : "ERROR"
    } ]
  }
}
```

Example response body (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<restMessageList>
  <list>
    <code>105</code>
    <message>Requested page number range string '1-5' does not contain valid
numbers</message>
    <type>ERROR</type>
  </list>
</restMessageList>
```

REST API reference v5

Preload PDF document with commands and prepare options


This request uploads PDF document and returns a reference ID for following delayed loading. The uploaded document is only available then in a temporary storage. In order to make it available for editing in the session workspace for a specific client he must load the uploaded document explicitly in another call (see [Activate preloaded PDF document for processing](#)) with the reference ID which was returned by the upload call.

If the clients wants to work directly with the document after uploading he can define the parameter `init=true`. With this parameter the uploaded document is loaded immediately in the session workspace of the requesting client for direct usage. In this case the preload call returns the document ID (instead of the reference ID).

In addition to the PDF document it is possible to define also commands and options for the prepare phase.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Content-Type: multipart/form-data

Method

POST

Example request

```
POST http://localhost:6610/sdweb/rest/v5/documents
```

Example request URL

```
POST http://localhost:6610/sdweb/rest/v5/documents
```

The following example shows a preload result with `init=false` and an included reference ID within the response body.

Response body (JSON)

Example preload result (`init=false`) with an included reference ID in the response body:

```
"restLoadId" : {
```

```
"type" : "REFID",  
"value" : "1393231005996_1dea12f6-e263-4f43-a144-594b1bb78112"  
}
```

Response body (XML)

Example preload result (init=true) with an included document ID in the response body:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
  <restLoadId xmlns="http://www.kofax.com/ksd/sdweb/rest/v5" >  
    <type>DOCID</type>  
    <value>123</value>  
  </restLoadId>
```

Query parameters

- **init** (boolean, optional): Causes the uploaded document to be available in the session workspace for the client on the server.
Other parameters (commands and options) can be sent in the BodyPart in the Multipart Message (see http://www.w3.org/Protocols/rfc1341/7_2_Multipart.html). Default value: false

The PDF document (part) can be sent either binary (as is) or as base64 encoded text.

The following **document data body** parts are supported:

- **docdata** (byte[] or string, required): The PDF document, either binary with media type application/octet-stream (of body part) or base64 encoded (with **media type: text/plain**)

Options and Commands (see also chapter [Integration in existing web applications](#)).

The following **Options** are supported:

- **docid** (string, optional): The ID of document.
- **format** (string, optional): Possible values are "PDF", "MS_WORD", "JPG", "JPEG", "PNG", "BMP" and "TIFF".
Default value: PDF. This means that PDF is the default document format.
- **clearsignatures** (string, optional): If the value is set to "true" then all signatures of the document are cleared.
- **originaltagid** (string, optional): If the value is set to "false" then the curly braces from the signature line tag will be removed when using it as signature name in the PDF document. Signature lines can be used to insert signature fields if the input document has format 'MS_WORD'.
- **dmsid** (string, optional): ID of DMS plugin
- **preparepluginid** (string, optional): The ID of Prepare plugin.
- **signaturearchiveid** (string, optional): The ID of SignatureArchive plugin.
- **validatepluginid** (string, optional): The ID of Validate plugin.
- **resultparamspluginid** (string, optional): The ID of ResultParams plugin.
- **docidsalt** (string, optional): Salt value for document ID generation, will be passed to getNewDocumentId(docidsalt) of DMS plugin (but only if docid parameter is not set and the document ID was not already defined before)
- **cmd[<uniqueid e.g. number>]** (string, optional): Command (see also chapter [Integration in existing web applications](#))

Commands

Example command with key and value for inserting text:

Key

cmd_1

Value

```
type=addtext|text>Lorem ipsum|pages=1|left=10|bottom=10|fontsize=100|  
fontname=Helvetica|textcolor=#FF0000|opacity=0.5
```

Example command with key and value for inserting a signature field.

Key

cmd_2

Value

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

Response

Status 201 (CREATED): The document could be preloaded. Otherwise a SignDoc Web status code is returned together with the explaining messages. If parameter `init` is set (true) then the response body returns the documentid of the preloaded document otherwise the reference ID which must be used for delayed loading of the document in session workspace.

- **restLoadId** (RestLoadId, required): Information which can be used for following document access. Load ID can be either document ID or reference ID.

RestLoadId


- **type** (RestLoadId.TYPE, optional):
Can be DOCID (if `init` parameter equals true) or REFID (if `init=false` for deferred loading)
- **value** (string, optional): ID value

Activate preloaded PDF document for processing

This request returns a snapshot of a previously loaded document which must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/refid/{refid}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Method

GET

Example request

GET http://localhost:6610/sdweb/rest/v5/documents/
refid/1393321111767_4acca35c-0974-440a-b632-9f5a7970f856

Example response body (JSON)

```
"restLoadId" : {  
  "type" : " DOCID ",  
  "value" : "123"  
}
```

Example response body (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<restLoadId xmlns="http://www.kofax.com/ksd/sdweb/rest/v5" >  
  <type>DOCID</type>  
  <value>123</value>  
</restLoadId>
```

Path parameters

- **refid** (string, required): Reference ID which was returned from a document preload request with init=false

Response status

Status 200 (OK): The requested document could be retrieved for download. Otherwise 404 (NOT FOUND), if ID not found or invalid. A SignDoc Web status code is returned together with an explaining message if the request was not successful. The response body returns the binary document, either in PDF format or as TIFF document.

Response status 200 (OK): The document could be activated for processing. Otherwise a SignDoc Web status code is returned together with the explaining messages.

The response body returns the document ID which must be used for later access to the document. The document ID was either explicitly specified in the preload call or was implicit generated by the server.

- **restLoadId** (RestLoadId, required): Information which can be used for following document access. Load ID contains the document ID (in this case).

RestLoadId

- **type** (RestLoadId.TYPE, optional):

Document ID. Default value: DOCID


- **value** (string, optional): Id value

Append PDF document to previously loaded document

The previously (up-) loaded target PDF document must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{did}/addon`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Content-Type: text/plain

Cookie: JSESSIONID=...

Method

POST

Example request

POST `http://localhost:6610/sdweb/rest/v5/documents/4711/addon`

Path parameters

- **did** (string, required): Document ID of the already loaded target document to which the passed on PDF document should be appended.

The PDF document must be sent as base64 encoded text in the request body.

- (string, required): The PDF document to be appended (base64 encoded with media type: text/plain)

Response status


Status 201 (CREATED): The document could be appended to the target document which is referenced by document ID. Otherwise a SignDoc Web status code is returned together with the explaining messages.

Attach PDF document to previously loaded document

The previously (up-) loaded target PDF document must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{did}/attachment/{name}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Content-Type: text/plain

Cookie: JSESSIONID=...

Method

POST

Example request

```
POST http://localhost:6610/sdweb/rest/v5/documents/4711/attachment/AuditTrail.pdf
```

Path parameters

- **did** (string, required): Document ID of the already loaded target document to which the passed on document should be attached.
- **name** (string, required): The name is used as filename of the attachment and must not contain slashes, backslashes, and colons. An attached pdf document must be named with the extension "pdf" otherwise Adobe Reader is not able to open or to export the extension. Example for a valid name is "AuditTrail.pdf"

Query parameters

- **description** (string, optional): Description text of the attachment (can be displayed in Adobe Reader)
- **lastmodification** (string, optional): The time and date of the last modification of the file being attached to the document (can be displayed in Adobe Reader). Must be in ISO 8601 extended calendar date format with optional timezone, e.g. 2009-06-30T18:30:00+02:00.

The PDF document must be sent as base64 encoded text in the request body.

- (string, required): The document to be appended (base64 encoded with media type: text/plain)

Response status


Status 201 (CREATED): The document could be attached to the target document which is referenced by document ID. Otherwise a SignDoc Web status code is returned together with the explaining messages.

Remove attachment from previously loaded document

The previously (up-) loaded target PDF document must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{did}/attachment/{name}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Cookie: JSESSIONID

Method

DELETE

Example request

```
DELETE http://localhost:6610/sdweb/rest/v5/documents/4711/attachment/
AuditTrail.pdf
```

Path parameters

- **did** (string, required): Document ID of the already loaded target document from where the attachment should be removed.
- **name** (string, required): The name of the attached document that should be removed. It is also used as filename of the attachment and must not contain slashes, backslashes, and colons.

Response status


Status 200 (OK): The attachment could be removed from the document which is referenced by document ID. Otherwise a SignDoc Web status code is returned together with the explaining messages.

Get document information

This request returns information about a previously loaded document, which must be available in the same session as the requested sessionid.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}/info`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

XML, JSON

Header

Accept: application/xml, application/json

Method

GET

Example request

```
GET http://localhost/sdweb/rest/v5/documents/123/info?
fields=text,capture&pages=1-3&metadata=true
```

Headers

Accept: Application/xml

Content-Type text/plain

Path parameters

- **docId** (string, required): Document ID (which was returned from a document preload request with `init=true`)

Query parameters

- **fields** (string, optional): Specifies fields for which field information should be returned. "all" is the default value and means all fields (of supported field types) are included within corresponding pages. You can define also subset of specific field types, separated by comma. Possible values are "capture" (includes captured signatures, but also images from a camera and Click2Sign fields),"text", "checkbox", "radiobutton". You could also provide the value "none" (as single value) if no field information is required. Default value: all
- **pages** (string, optional): Specifies the pages information that should be included. "all" is the default value and means all pages of the document. You can define a single page number, a list of page numbers (separated by commas) or a range of page numbers (separated by a minus sign '-') or "0" if no page info is required. Examples are '1, 2,4' or '2-5' or also '1-3,5' (without quotes). Default value: all

- **metadata** (boolean, optional): Specifies whether SignDoc Web specific metadata should be included in the response. SignDoc Web metadata is stored in the 'encrypted' collection of the SignDoc document properties. Default value: true

Response status

Status 200 (OK): The document information could be retrieved. Otherwise a SignDoc Web status code is returned together with the explaining messages.

The response body returns a RestDocumentOutput structure.

RestDocumentOutput

- **signatureFields** (List<RestSignatureFieldOutput>, optional): List of signature field information elements. One entry with information is included in the list for each signature field within the document. Precondition for this list is that signature fields are requested, either by setting the fields parameter to 'all' (or by omitting the fields parameter, which is the same as setting 'all') or by setting the fields parameter to signature (as one of the allowed parameter values, separated by comma).
- **textFields** (List<RestTextFieldOutput, optional): List of text field information elements. One entry with information is included in the list for each text field within the document. Precondition for this list is that text fields are requested, either by setting the fields parameter to 'all' (or by omitting the fields parameter, which is the same as setting 'all') or by setting the fields parameter to 'text' (as one of the allowed parameter
- **checkboxFields** (List<RestCheckboxFieldOutput, optional): List of checkbox field information elements. One entry with information is included in the list for each checkbox field within the document. Precondition for this list is that text fields are requested, either by setting the fields parameter to 'all' (or by omitting the fields parameter, which is the same as setting 'all') or by setting the fields parameter to 'checkbox' (as one of the allowed parameter values, separated by comma
- **radioButtonFields** (List<RestRadioButtonFieldOutput>, optional): List of radio button (group) field information elements. One entry with information is included in the list for each radio button (group) field within the document. Precondition for this list is that text fields are requested, either by setting the fields parameter to 'all' (or by omitting the fields parameter, which is the same as setting 'all') or by setting the fields parameter to 'radiobutton' (as one of the allowed parameter values, separated by comma).
- **id** (string, optional): The document ID (must be unique within the session)
- **description** (string, optional): The document ID (at present)
- **totalPageNumber** (integer, optional): The total number of pages in the document
- **pages** (List<RestPageInfo, optional): A list of page information elements, one entry for each included page. The pages parameter determines whether which page information is included in the list.
- **metaDataList** (RestMetaDataSet, optional): A list of key/value elements with 'metadata' information of the document.

RestPageInfo

- **description** (string, optional): Includes currently only page number information
- **number** (string, optional): Page number within the document (1-n)
- **width** (number, optional): Width of the page

- **height** (number, optional): Height of the page
- **tooltip** (string, optional): Includes currently only page number information
- **url** (string, optional): The URL for requesting the page image
- **conversionFactorX** (double, optional): Get the horizontal conversion factor for a page. Different pages of the document may have different conversion factors. Divide horizontal coordinates by the returned number to convert document coordinates to inches. The return value will be 0.0 if the factor is not available
- **conversionFactorY** (double, optional): Get the vertical conversion factor for a page. Different pages of the document may have different conversion factors. Divide horizontal coordinates by the returned number to convert document coordinates to inches. The return value will be 0.0 if the factor is not available

RestMetaDataList

- **list** (List<RestMetaData>, optional): List of metadata elements

RestMetaData

- **key** (string, optional): The key of the metadata entry
- **value** (string, optional): The value of the metadata entry

RestSignatureFieldOutput

- **captureFieldSubtypeChoice** (List<ERestCaptureFieldSubtype, optional): A list of selectable signature subtypes. The capture field subtype choice can contain one or one capture subtype definitions:
CFST_SIGNATURE if the signer can sign with a pad device or via mouse or pen.
CFST_C2SSIGNATURE if the field can be signed by entering a text as "Click-to-Sign" signature.
CFST_IMAGECAPTURE if the signer can sign via captured photo from a camera.
- **captureFieldSubtype** (ERestCaptureFieldSubtype, optional): The capture field subtype. This can be CFST_SIGNATURE if the field contains a signature. It is CFST_C2SSIGNATURE if the field was signed with a "Click-to-Sign" signature. It contains CFST_IMAGECAPTURE if the signature is based on a captured image (via camera). If the field is not (yet) signed, the captureFieldSubtype returns CFST_UNKNOWN
- **signed** (boolean, optional): This flag informs whether the signature field is signed or not.
- **usesLock** (boolean, optional): Returns true if the signature field causes any field locks after signing.
- **imageWidth** (integer, optional): The calculated or defined image width if captureFieldSubtypeChoice contains CFST_IMAGECAPTURE. Default value: 240
- **imageHeight** (integer, optional): The calculated or defined image height if captureFieldSubtypeChoice contains CFST_IMAGECAPTURE. Default value: 320
- **imageUrl** (string, optional): The URL for retrieving the image snippet of the field.

Additionally see table "The base definitions of all fields".

RestTextFieldOutput

- **value** (string, optional): The value of the text field
- **multiLine** (boolean, optional): Indicates whether the field is a single line or a multiline text field.

- **maxLength** (integer, optional): The maximum length of the text field value. If the maxLength element is not included, the text field does not have a maximum length.

Additionally see table "The base definitions of all fields".

RestCheckboxFieldOutput

See table "The base definitions of all fields" and especially the field attributes for checkboxes and radio button (group) fields in the RestWidget structure, first of all the attribute 'selected'.

The base definitions of all fields

- **name** (string, optional): The field name.
- **required** (boolean, optional): The flag indicates whether the field is mandatory (required=true) or optional.
- **readOnly** (boolean, optional): The flag indicates whether the field can be changed (readOnly=false) or not.
- **alternateName** (string, optional): An alternate name for the field can be set, e.g. as more readable name or as label in the client. If no alternate name is set for the field, this element could also contain the 'friendly' name of the field (value of metadata SIGNDOCWEB_FRIENDLYNAME_ + fieldname).
- **tooltip** (string, optional): A client side used tooltip can be defined for each field. It is stored in the metadata SIGNDOCWEB_TOOLTIP_ + fieldname.
- **widgets** (List<RestWidget>, optional): A list of widget objects. A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, i.e, widgets.

RestWidget

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).
- **pageNumber** (integer, optional): The page number within the document.
- **top** (double, optional): Set the top coordinate. The origin is in the bottom left corner of the page. See [Document coordinate system](#).
- **left** (double, optional): Set the left coordinate. The origin is in the bottom left corner of the page. See [Document coordinate system](#).
- **right** (double, optional): Set the right coordinate. The origin is in the bottom left corner of the page. See [Document coordinate system](#).
- **bottom** (double, optional): Set the bottom coordinate. The origin is in the bottom left corner of the page. See [Document coordinate system](#).
- **imageUrl** (string, optional): The URL for requesting the field related image snippet.
- **locked** (boolean, optional): The flag locked=true indicates that the widget cannot be deleted or modified, but its value can be changed.

Field attributes for text fields

- **textJustification** (ERestTextJustification, optional): The justification for text fields. Can be LEFT, CENTER or RIGHT.

- **fontName** (string, optional): The font name of the text field.
- **fontSize** (double, optional): 0.0 means auto size of text field font
- **textColor** (RestColorRGB, optional): The RGB (Red, Green, Blue) color value of the text. An RGB value is specified with: rgb(red, green, blue).
Each parameter (red, green, and blue) defines the intensity of the color as an integer between 0 and 255.
For example, rgb(0, 0, 255) is rendered as blue, because the blue parameter is set to its highest value (255) and the others are set to 0.
No text color is specified if the attribute textColor is omitted.

Field attributes for checkboxes and radio button (group) fields

- **selected** (boolean, optional): Indicates whether the checkbox or a specific radio button (widget) is selected or not.
- **buttonValue** (string, optional): For radio button fields and check box fields, each widget also has a "button value". The field proper has a value which is either "Off" or one of the button values of its widgets.
Each widget of a radio button field or a check box field is either off or on. If all widgets of a radio button field or a check box are off, the field's value is "Off". If at least one widget is on, the field's value is that widget's "button value". As the value of a field must be different for the on and off states of the field, the button values must not be "Off".

RestColorRGB

- **red** (integer, optional): The intensity of the red color, 0-255
- **green** (integer, optional): The intensity of the green color, 0-255
- **blue** (integer, optional): The intensity of the blue color, 0-255

The following example shows a response JSON body with requested pageInfoList and include MetaData list.

Example response body (JSON)

```
{
  "restDocumentOutput" : {
    "id" : "f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a",
    "description" : "f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a",
    "pageTotalNumber" : 1,
    "pages" : [ {
      "number" : 1,
      "width" : 595.3200073242188,
      "height" : 841.9199829101562,
      "description" : "Page 1",
      "tooltip" : "Page 1",
      "url" : "http://localhost:80/sdweb/rest/v5/documents/f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a/pages/1/image?spts=1521550395043",
      "conversionFactorX" : 72.0,
      "conversionFactorY" : 72.0
    } ],
    "metaDataList" : {
      "list" : [ {
        "key" : "SIGNDOCWEB_INTERNAL_DOCUMENT_ID",
        "value" : "f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a"
      }, {
        "key" : "SIGNDOCWEB_FRIENDLYNAME_signature-1",
```

```

        "value" : "signer 1"
      }, {
        "key" : "SIGNDOWEB_INTERNAL_RESULTPARAMS_PLUGIN_ID",
        "value" : "de.softpro.sdweb.plugins.impl.DefaultResultParams"
      }, {
        "key" : "signature-1_SUBTYPE",
        "value" : "capture"
      }, {
        "key" : "SIGNDOWEB_CAPTURE SUBTYPES_CHOICE_signature-1",
        "value" : "CFST_C2SSIGNATURE,CFST_SIGNATURE,CFST_IMAGECAPTURE"
      }, {
        "key" : "SIGNDOWEB_REQUIRED_signature-1",
        "value" : "true"
      }
    ]
  },
  "signatureFields" : [ {
    "name" : "signature-1",
    "required" : true,
    "readOnly" : false,
    "alternateName" : "",
    "widgets" : [ {
      "index" : 0,
      "pageNumber" : 1,
      "top" : 834.0,
      "left" : 8.0,
      "right" : 158.0,
      "bottom" : 786.0,
      "imageUrl" : "http://localhost:80/sdweb/rest/v5/documents/
f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a/pages/1/image?
top=834.0&left=8.0&bottom=786.0&spts=1521550395043&zoomfactor=100&right=158.0",
      "locked" : false
    } ],
    "captureFieldSubtypeChoice" : [ "CFST_C2SSIGNATURE", "CFST_SIGNATURE",
"CFST_IMAGECAPTURE" ],
    "captureFieldSubtype" : "CFST_UNKNOWN",
    "signed" : false,
    "imageWidth" : 320,
    "imageHeight" : 240
  } ]
}
}

```

Example response body (XML)

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<restDocumentOutput xmlns="http://www.kofax.com/ksd/sdweb/rest/v5">
  <id>f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a</id>
  <description>f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a</description>
  <pageTotalNumber>1</pageTotalNumber>
  <pages>
    <number>1</number>
    <width>595.3200073242188</width>
    <height>841.9199829101562</height>
    <description>Page 1</description>
    <tooltip>Page 1</tooltip>
    <url>http://localhost:80/sdweb/rest/v5/documents/f6305e93_-ff3c76ba-b650-4c12-
a7d8-0af248cbb20a/pages/1/image?spts=1521550395043</url>
    <conversionFactorX>72.0</conversionFactorX>
    <conversionFactorY>72.0</conversionFactorY>
  </pages>
  <metaDataList>
    <list>
      <key>SIGNDOWEB_INTERNAL_DOCUMENT_ID</key>
      <value>f6305e93_-ff3c76ba-b650-4c12-a7d8-0af248cbb20a</value>
    </list>
  </metaDataList>
</restDocumentOutput>

```

```

</list>
<list>
  <key>SIGNDOCWEB_FRIENDLYNAME_signature-1</key>
  <value>signer 1</value>
</list>
<list>
  <key>SIGNDOCWEB_INTERNAL_RESULTPARAMS_PLUGIN_ID</key>
  <value>de.softpro.sdweb.plugins.impl.DefaultResultParams</value>
</list>
<list>
  <key>signature-1_SUBTYPE</key>
  <value>capture</value>
</list>
<list>
  <key>SIGNDOCWEB_CAPTURE_SUBTYPES_CHOICE_signature-1</key>
  <value>CFST_C2SSIGNATURE,CFST_SIGNATURE,CFST_IMAGECAPTURE</value>
</list>
<list>
  <key>SIGNDOCWEB_REQUIRED_signature-1</key>
  <value>true</value>
</list>
</metaDataList>
<signatureFields>
  <name>signature-1</name>
  <required>true</required>
  <readOnly>false</readOnly>
  <alternateName/>
  <widgets>
    <index>0</index>
    <pageNumber>1</pageNumber>
    <top>834.0</top>
    <left>8.0</left>
    <right>158.0</right>
    <bottom>786.0</bottom>
    <imageURL>http://localhost:80/sdweb/rest/v5/documents/
f6305e93_ff3c76ba-b650-4c12-a7d8-0af248cbb20a/pages/1/image?
top=834.0&left=8.0&bottom=786.0&spts=1521550395043&zoomfactor=100&
right=158.0</imageURL>
    <locked>false</locked>
  </widgets>
  <captureFieldSubtypeChoice>CFST_C2SSIGNATURE</captureFieldSubtypeChoice>
  <captureFieldSubtypeChoice>CFST_SIGNATURE</captureFieldSubtypeChoice>
  <captureFieldSubtypeChoice>CFST_IMAGECAPTURE</captureFieldSubtypeChoice>
  <captureFieldSubtype>CFST_UNKNOWN</captureFieldSubtype>
  <signed>false</signed>
  <imageWidth>320</imageWidth>
  <imageHeight>240</imageHeight>
</signatureFields>
</restDocumentOutput>


```

Get document page image

This request returns an image of a document page of a previously loaded document which must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}/pages/{pageNo}/image/{format}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

image/png, image/jpeg, image/gif, image/bmp, image/tiff

Header

Accept: image/png, image/jpeg, image/gif, image/bmp, image/tiff (optional, see description of path parameter format), application/json, application/xml

Cookie: JSESSIONID

Method

GET

Example request

GET http://localhost:6610/sdweb/rest/v5/documents/123/pages/1/image

Path parameters

- **docId** (string, required): Document ID (which was returned from a document preload request with `init=true`)
- **pageNo** (string, required): Requested page number of the document
- **format** (string, optional): Image format, can be defined alternatively to Accept-Header (if both is defined then format has higher priority). Possible format values are png, jpeg, gif, bmp or tiff.

Query parameters

- **zoomfactor** (integer, optional): Requested zoom factor in percent. Minimum value is 25 (%), maximum value is 200 (%). Default value: 100
- **top** (number, optional): It is possible to request only a page snippet of a document. The position of the image is defined with the x and y document coordinates (*) of a rectangle within the specified page. The snippet can be rendered only if all coordinates (top, bottom, left, right) are provided in the request. The top parameter value which describes the upper y-coordinate must be greater than the bottom value.
- **bottom** (number, optional): This value is lower y-coordinate of the requested page snippet (see also parameter top).
- **left** (number, optional): The left parameter defines the left x-coordinate of the requested page snippet (see also parameter top) and must not be equal or greater than the right value.
- **right** (number, optional): The right parameter specifies the right x-coordinate of the requested page snippet (see also parameter top) and must be greater than the value of the parameter left.

* The origin of the document coordinate system is in the bottom left corner of the page (as rendered, that is, taking rotation of PDF pages into account). Points having positive X coordinates are to the right of the origin, points having positive Y coordinates are above the origin.

For PDF documents, the origin is in that corner of the intersection of the CropBox and the MediaBox of the page that corresponds to the bottom left corner of the image that would be rendered for that page. The units are specified by the PDF document and are usually 1/72 inch.

For TIFF documents, the origin is in the bottom left corner of the page, the unit is one pixel.

Response status

Status 200 (OK): The page image could be rendered. Otherwise a SignDoc Web status code is returned together with an explaining message.


The response body returns the binary page image in requested format. The request can fail if the megapixels of the requested image are less than the value specified by the "sdweb.document.image.max_mp" configuration property. The default is a maximum of 10 megapixels.

Get document

This request returns a snapshot of a previously loaded document which must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

application/pdf, image/tiff

Header

Accept: application/pdf, image/tiff, application/json, application/xml

Cookie: JSESSIONID=...

Method

GET

Example request

GET `http://localhost:6610/sdweb/rest/v5/documents/123?flatten=true`

Path parameters

- **docId** (string, required): Document ID (which was returned from a document preload request with `init=true`)

Query parameters

- **flatten** (boolean, optional):

If set to true all fields are flattened (no more fields are available which could be accessed or even changed only the appearance of the last field state is visible in the document). This is only supported for PDF documents.

- **signid** (string, optional): It is possible to create a document snapshot at the capture time of a specific signature. The signid specifies the name of the signature field for this specific signature.
- **content_disposition** (string, optional):

There are situations (when downloading a PDF document) where you might want a hyperlink leading to a file to present a SaveAs dialog in browser. This could (browser dependent) be reached by setting the response header Content-Disposition: attachment; filename="<file name.ext>".

The query parameter content_disposition sets this Content-Disposition header value in the response. Usually "attachment" and "inline" are supported by a browser (see also <http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html> - 19.5.1 Content-Disposition

Response status


Status 200 (OK): The requested document could be retrieved for download. Otherwise 404 (NOT FOUND), if ID not found or invalid. A SignDoc Web status code is returned together with an explaining message if the request was not successful. The response body returns the binary document, either in PDF format or as TIFF document.

Get audit logs of document

This request gets all accumulated Audit Log entries from a previously loaded document available in session workspace (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)). The response returns an Audit Trail specific XML structure (could contain BASE64 encoded binaries, e.g. images) of all current audit logs. The XML schema is available as AuditTrail.xsd in signdoc_xml_interfaces_x.x.jar (whereas x.x is the current version number).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}/auditlogs`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

XML

Header

Accept: application/json, application/xml

Cookie: JSESSIONID=...

Method

GET

Example request

```
GET http://localhost:6610/sdweb/rest/v5/documents/123/auditlogs
```

Path parameters

- **docid** (string, required): Document ID (was returned from a document preload request with `init=true`)

Response status


Status 200 (OK): The requested audit log entries could be successfully retrieved from the document. Otherwise the status is 404 (NOT FOUND), if ID not found or invalid. A SignDoc Web status code is then returned together with an explaining message.

Add signature

This request adds a signature to an existing signature field of a previously loaded document available in session workspace (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

```
http://host_server:port_number/sdweb/rest/v5/documents/{docid}/signaturefields/{fname}/signature/{sigtype}
```

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes

multipart/form-data

Produces

JSON, XML

Header

Accept: application/pdf, image/tiff, application/json, application/xml

Accept: application/json, application/xml

Content-Type: multipart/form-data

Method

POST

Example request

```
POST http://localhost/sdweb/rest/v5/documents/doc-1/signaturefields/sig-1/signature/SIGNWARE
```

Example Header

Accept: application/json

Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryNBRKRwormSsilT40

Cookie: JSESSIONID=5AD05ECF362772DA2846E72C1759515B

Path parameters


- **docId** (string, required): The ID of the related PDF document
- **fname** (string, required): Signature field name in the PDF document
- **sigType** (string, required): The signature type (*). It can have one of the following values: SIGNWARE, IMAGE_BMP_1BIT, IMAGE, C2S, SIGNATURE_A and SIGNATURE_B

Signature types (*)

- **SIGNWARE** The ID of the related pdf document
- **IMAGE_BMP_1BIT** Monochrome image in BMP format
- **C2S** Click-to-Sign image is created from (mandatory) body part signer_name parameter value
- **SIGNATURE_A** SignDoc Web internal signature format
- **SIGNATURE_B_A** SignDoc Web internal signature format

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

 The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to add a signature to a signature field in a SignDoc document the request must contain a valid X-S-AUTH-TOKEN header for authentication.

The following entries must be added in sdweb_config.groovy (for usage with SignDoc):

```
sdweb.plugins.loadlist <<  
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'  
  
sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

The following document data body parts are supported:

- **sigdata** (byte[] or string, optional (*)): The signature data, either binary with media type application/octet-stream (of body part) or base64 encoded (media type: text/plain). (*) the parameter is mandatory if sigType is not C2S
- **signer_name** (string, optional (**)): (**) The signer_name is mandatory if sigType=C2S is provided. The signer name is rendered as signature image for a click-to-sign capture field.
- **encoding** (string, optional): Describes the encoding of the provided signature in sigData. Supported values are
binary
base64

Base64 encoded string. For SIGNATURE_A and SIGNATURE_B signatures the first byte contains the qualifier for the format, 'a' for SIGNATURE_A and 'B' for SIGNATURE_B. This is also the case for a base64 encoded signature in one of these both formats. That means that the only the part after this preceding qualifier is base64 encoded

base64zip

Zip archived base64 encoded string

nibblehex

Hexadecimal encoded string

Default: If encoding parameter is not provided then either 'binary' is assumed if media type of form part sigData is "application/octet-stream" or 'base64' if media type of form part sigData is not "application/octet-stream"


- **commonName** (string, optional): CN signer name for self-signed (one-time) certificate (1024 bits). A one-time generated certificate is used for signing if no certificate is provided and if no (configured) default certificate is available.
- **esignkey** (string, optional): The public key for encrypting the biometric signature data (RSA). It is essential to encrypt biometric data asymmetrically and to keep the private key secret. To create the RSA key pair, you can use either JRE KeyTool (which will use a proprietary file format for encrypted private keys) or any tool that creates an RSA key pair and uses PKCS #1 format (DER or PEM) for the public key and PKCS #12 format for the private key. Alternatively, the public key can also be specified as X.509 certificate (DER or PEM).
- **signer_misc** (string, optional): Miscellaneous signer text which is passed to the click-to-sign signature renderer plugin (not considered in the default implementation DefaultC2SSignatureRenderer)
- **CFSTChoice** (string, optional): Important: Currently only supported in SignDoc Web without SignDoc environment.
One or more possible capture field subtype definitions (separated by comma) which can be used as input for a signature field. This choice is provided to the user in the client as possible capture methods if he clicks on a signature. Allowed values are c2s (for click-to-sign), signature or image_capture.
Example: c2s,signature,image_capture
Default: Either from metadata SIGNDOWEB_CAPTURE_SUBTYPES_CHOICE_ + fieldname, if available, or the derived (single) value from the path parameter sigtype, either signature (for sigtype SIGNWARE, SIGNATURE_A or SIGNATURE_B) or image_capture (for sigtype IMAGE_BMP_1BIT or IMAGE) or c2s for sigtype C2S.
- **signing_certificate_key** (string, optional): Important: Currently only supported in SignDoc environment, not with SignDoc Web only.
The signing_certificate_key is a RSA encrypted AES key (base64 encoded) which is necessary for decryption of the encrypted certificate and the encrypted password if provided.
Note: The initial vector (IV) for the AES encryption and the public key for RSA encryption (of the AES key) can be retrieved by GET /v5/configuration request. The IV is provided as value for the key client.signing.iv. The public key is available as value for the key client.signing.pubkey in PEM (Privacy enhanced Mail) format (base64 encoded).
- **signing_certificate_encrypted** (string, optional): Important: Currently only supported in SignDoc environment, not with SignDoc Web only.
AES encrypted certificate (PKCS#12 base64 encoded) which should be used for signing. The corresponding AES key must be provided with the parameter signing_certificate_key.

- **signing_certificate_pass_encrypted** (string, optional): Important: Currently only supported in SignDoc environment, not with SignDoc Web only!.
AES encrypted certificate password (base64 encoded), if required for provided certificate. The corresponding AES key must be provided with the parameter `signing_certificate_key`.
- **signing_certificate** (string, optional): Unencrypted certificate (PKCS#12 base64 encoded) which should be used for signing.
- **user_pkcs12** (binary, optional): Unencrypted certificate (PKCS#12 as binary) which should be used for signing.
- **user_certificate_password** (string, optional): Unencrypted certificate password (as clear text), if required for provided certificate for signing.
- **user_certificate** (string, optional): The file name of the PKCS#12 certificate with ".p12" suffix from %SDWEB_HOME%/conf/user_certificates which should be used for signing:
Example: If you want to use %SDWEB_HOME%/conf/user_certificates/MyCert.p12 you must provide `user_certificate=MyCert` as parameter.

Response

Status 201 (CREATED): The signature field was successfully signed.

The body of the response contains additional information in the `RestAddSignatureResult` structure (JSON or XML).

 Calls to a Signature Archive plugin which can be defined via `signaturearchiveid` in the 'Upload document' request are supported in this request.

Response body - `restAddSignatureResult`

Path parameters

- **resultCode** (string, optional): The result code could have one of the following values (see also table "Result codes"):
SUCCESS, SIGNATURE_TOO_SIMPLE_OR_NOT_USABLE, SA_MATCH, SA_NO_MATCH, SA_NOT_FOUND, SA_GENERIC_ERROR, SA_QUESTIONABLE_MATCH
- **fieldsToUpdate** (List<RestField>, optional): Contains a list of RestField objects of those fields which has been changed after adding a signature to a signature field. Currently only the changed signature field itself is included.
- **addSignatureResults** (Map<String, String>, optional): A list of possible result attributes from the signature archive plugin. Possible entries are:
GUI_MESSAGE
The message, that could be displayed on the GUI if the validation result from the archive plugin is VALIDATION_RESULT is QUESTIONABLE_MATCH or NOT_FOUND
HIDDEN_PARAMETER
A hidden parameter, the plugin can use
REFERENCE_ID
The reference ID of the signature within the session workspace if the archive plugin returns a questionable match (QUESTIONABLE_MATCH)
REFERENCE_SIG_IMAGE_URL

The URL for retrieving the reference signature

TEST_SIG_IMAGE_URL

The URL for retrieving the reference signature

Result codes

- **SUCCESS** The signature could be successfully added to the signature field.
- **SIGNATURE_TOO_SIMPLE_OR_NOT_USABLE** If sigtype is not C2S then a signature must be provided in the body in form parameter sigdata. This result code is returned if no sigdata element is provided for this caseSignDoc
- **SA_MATCH** Result from the SignatureArchive plugin
- **SA_NO_MATCH** Result from the SignatureArchive plugin
- **SA_NOT_FOUND** Result from the SignatureArchive plugin
- **SA_GENERIC_ERROR** Result from the SignatureArchive plugin
- **SA_QUESTIONABLE_MATCH** Result from the SignatureArchive plugin

RestField structure

- **name** (string, optional): The name of the field within the document.
- **type** (string, optional): The field type, can be FT_TEXT, FT_CHECKBOX, FT_RADIOBUTTON or FT_CAPTURE
- **mandatory** (string, optional): Defines whether the field is a required field in the document
- **readOnly** (string, optional): Defines whether the field is a read only field in the document which cannot be changed.
- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **tooltip** (string, optional): The tooltip of the field could be also displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_TOOLTIP_" + (field) name.
Example: SIGNDOCWEB_TOOLTIP_sig1
- **widget** (string, optional): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, i.e, widgets.

RestWidget structure

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).
- **pageNumber** (integer, optional): The page number within the document.
- **top** (string, optional): The top coordinate of the field in the page. The origin is in the bottom left corner of the page. See [Document coordinate system](#).
- **left** (string, optional): The left coordinate of the field in the page. The origin is in the bottom left corner of the page. See [Document coordinate system](#).
- **right** (string, optional): The right coordinate of the field in the page. The origin is in the bottom left corner of the page. See [Document coordinate system](#).

- **bottom** (string, optional): The bottom coordinate of the field in the page. The origin is in the bottom left corner of the page. See [Document coordinate system](#).

Example request

POST http://beaker:8081/sdweb/rest/v5/documents/8b055a9d_-_document-1/signaturefields/aealf1a7-8024-4f45-8455-9a007f8dd9d5/signature/SIGNWARE

Example header

Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryn1Wrf9HtKDXM2Qex

Content-Length: 2330

Accept: application/json, text/plain, */*

Cookie: JSESSIONID=D0204F744BE26DAF7AC895240337037B

Example request body (multi part form data)

```
-----WebKitFormBoundaryn1Wrf9HtKDXM2Qex
Content-Disposition: form-data; name="sigdata"
```

98191107390A0000F8030000789C75D67F6895551807F073CF39AFF7C7F6D64C0925CD490E91B2A0A4FC
63194A5A66B939078A8D68E05814692E3051318758A4242A4B44448D2815A59659991A1232344283B524
2A0A23A184C2D608B152BFCC79BEB773881ADC9DCF1DEF3DE7739CF79303030303030303033A1C94CA78B3C
762C6E269CC043BDFE1E138DBC87646FA140B076F7290CD9560000000000000000AECBE7AC74F170FC
D27647F04A83EB87BD6A6381D52726F23A49ED5E6DB730C4EAD9269046BA2E7AC8FBEE96FCF1C9D3EA
BF8D455 69B41B1008AA398065CBD2D601E754C87A2C4D01771EFCDD004D6BD54C8924D957862B9ED0C33
39F7DED01496B97837B78586ABFFE24C2DB55713FB7DAAB3DA38CE92D741814C75A029BE72F5A4BFDB23
2742C89AD7D33C4E898CDD9ADE1121B2752CE0832A0B594ECD388C3763A4041FE46B7E10B9EAEEDEBE9F4
0E1B50FB0E71D45F5A3DEAD3A4FFE4310BF4B1BC19C54E1DB08B0299F012FF617FA80E8682E982E137EF
3F2773F087BA1C352981E104848B72E2B58073DA43741FCE7F0087811FC097C1BD0C01FA647E95AF66E3
D3BDF4E63EF1E35686760FC1A319F575B9C99439E2CF264F0373781369A37DB4E7FFF2F2FFB1F77D3D89
FDD4C0FD287F45CEC1EEC73906BC1F637FA1D8CFDD17E3B7D8279AE576753390F7291B56A1D561DD6FD8
E3E93781F5EBFD25B12E36EDAABF44A9D37EBD43DB9B1F0B6E8503FB83B0EFD62D8CFBA96B888F374C8B
1C3DD2AA1B6DC5AB5D53A9DF7287C0AE345B541A9FC18FC3D88B1F454EF7C7C0FC8C683724E075
E05A33BEDEC7F039F9F4EAD04923BEF21F8933FA6F7C35D70235DE0A1F69CFF115FA84D6B0FB54ED7E4F8
C7EE9BED2DE227388C31E65BF3FC1725EB3D595B2E6D25D86D1636CBFBA06BDD2BE0BE33C6BDAB4FF04F
7686F0C3E0AAFA34F263E9778303E5F9BE9F906DFAEFD2A78116310A357BA167A27C6AD89FBA2C3B954D
D40CF83DBE25A7E698CDFAF5057503BFE2575F970E291D125C4E5BB69E4D03FCD6726F82CAA9F1DC1396
5FD0BB4D4D1875C1739749B68C977172D712DA0659D07E95D18EFA37BF5A059FD17E5BCD673BF1A1C29D
A1382254F6ABF33CE627DF1C737283D4520FBD5D7B94F846C49F591ADF5B591D2D7D7634BD35F1A1C46
713FF1D3F5B27FD738896FCFF402F89EBD621167F20714FE2A5C9671FA1D137FD78AEF59CF6C6B097D73
1BEC7FDCABD7F25D6987B329E7B356FA526ED51E2EC6AB4EFA7673E359D823F1BC2C7ABB5B9F7853E2F
DB4AC7F9C3DC8CD717F4E5E8D00FE9F0BD4917B74697F03DEFCEB2DE56637C937B9913D7AD48AF5EC5D
A407E5C078DF83A45705B724F526B5F0585287556F4CBC37DEC1506F9CA7765A9C3FD45BB5F606125F8
ACE65EF6B13AF4FCCF873C937EF782EBD794F62E6336F4C6ABB31DEAFE090E7EB6E0B75E0

```
-----WebKitFormBoundaryn1Wrf9HtKDXM2Qex
Content-Disposition: form-data; name="encoding"
```

NIBBLEHEX

```
-----WebKitFormBoundaryn1Wrf9HtKDXM2Oex--
```

Response

Status code 201 (OK): The signature could be successfully added.

Status code 404 (NOT FOUND): The document is not available in the requested session.

In case of status code 201 the response body contains the “add signature result” structure `restAddSignatureResult`

Example of response body

```
{
  "restAddSignatureResult" : {
    "resultCode" : "SUCCESS",
    "fieldsToUpdate" : [ {
      "name" : "signature-1",
      "type" : "FT_CAPTURE",
      "mandatory" : false,
      "readOnly" : false,
      "widgets" : [ {
        "index" : 0,
        "pageNumber" : 1,
        "top" : 400.0,
        "left" : 100.0,
        "right" : 400.0,
        "bottom" : 300.0,
        "locked" : true,
        "fontName" : "Helvetica",
        "fontSize" : 8.0,
        "textColor" : {
          "red" : 128,
          "green" : 128,
          "blue" : 128
        }
      } ],
      "captureFieldSubtype" : "CFST_SIGNATURE",
      "captureFieldSubtypeChoice" : [ "CFST_C2SSIGNATURE, CFST_SIGNATURE" ],
      "signed" : true
    } ],
    "addSignatureResults" : {
      "entry" : [ ]
    }
  }
}
```

Insert signature field

This request inserts a signature field into a pdf document.

i The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/signaturefield`

i *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes and produces

JSON, XML

Header

Accept: application/json, application/xml

Method

POST

Example request

```
POST http://localhost:6610/sdweb/rest/v5/documents/42/signaturefield
```

Example header


```
Accept: application/json
```

Path parameters

- **docid** (string, required): The ID of the related pdf document

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

 The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to insert a field in a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'

sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

Body input

The body contains the `RestSignatureFieldInput` structure in JSON or XML format.

The specification has to provide at least the field name and one widget definition with values for left, right, top, bottom and pageNumber.

Body parameters in RestSignatureFieldInput structure

- **captureFieldSubtypeChoice** (List<ERestCaptureFieldSubtype>, optional): A signature subtype can be a signature, an image or a “click to sign” (c2s) signature. The input is a comma separated list of signature subtypes which should be offered to a signer for signing a signature field.
Possible input values: CFST_SIGNATURE, CFST_C2SSIGNATURE, CFST_IMAGECAPTURE
- **id** (string, optional): The ID of the field in SignDoc.
- **name** (string, optional): The name of the field, must be unique within the document

- **signerId** (string, optional): If the signature should be captured by a specific person, the ID of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDOCWEB_FIELD_SIGNER_sig1
- **signerName** (string, optional): If the signature should be captured by a specific person, the name of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDOCWEB_FIELD_SIGNER_sig1
- **required** (boolean, optional): Defines whether the new signature field is a required field in the document which must be signed or if capturing of a signature is optional. Default is optional. Allowed values are true or false.
Default value: false
Note: The flags required and readOnly cannot be set to true for the same field.
- **readOnly** (boolean, optional): Defines whether the new signature field is a read only field in the document which cannot be signed. Default is signable. Allowed values are true or false.
Default value: false
Note: The flags required and readOnly cannot be set to true for the same field.
- **value** (string, optional): The value of the text field.
- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **tooltip** (string, optional): The tooltip of the field could be also displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_TOOLTIP_" + (field) name.
Example: SIGNDOCWEB_TOOLTIP_sig1
- **widgets** (List of RestWidget objects, required): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, ie, widgets.

RestWidget structure

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).
- **pageNumber** (integer, required): The page number within the document.
- **top** (double, optional): Set the top coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **left** (double, required): Set the left coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **right** (double, required): Set the right coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **bottom** (double, required): Set the bottom coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).

Example request

POST http://localhost:6610/sdweb/rest/v5/documents/42/signaturefield?
packageid=package1&fid=a2348a11-294b-4336-c11a-aaa34590887d

Example request body (JSON)

```
{
  "restSignatureFieldInput": {
    "captureFieldSubtypeChoice": ["CFST_SIGNATURE", "CFST_C2SSIGNATURE"],
    "name": "signature-1",
    "required": false,
    "alternateName": "Signature field 1",
    "tooltip": "This is signature field number 1",
    "widgets": [
      {
        "index": 0,
        "pageNumber": 1,
        "left": 511.12,
        "top": 120.02,
        "right": 591.12,
        "bottom": 80.12
      }
    ]
  }
}
```

Response status

Status code 201 (OK): The field could be successfully inserted.

Status code 404 (NOT FOUND): The document was not available in the requested session.

In case of status code 201 the response body contains the complete definition of the inserted signature field in a RestDocumentOutput structure.

Examples of response body

```
{
  "restDocumentOutput" : {
    "id" : "signdocbdo_-_022fda5b-acc3-8825a-8348-a3e456afd9c4",
    "signatureFields" : [ {
      "captureFieldSubtypeChoice": ["CFST_SIGNATURE", "CFST_C2SSIGNATURE"],
      "name" : "signature-1",
      "required" : false,
      "readOnly" : false,
      "alternateName" : "Signature field 1",
      "tooltip" : "This is signature field number 1",
      "widgets" : [ {
        "index" : 0,
        "pageNumber" : 1,
        "left": 511.12,
        "top": 120.02,
        "right": 591.12,
        "bottom": 80.12
      } ]
    } ]
  }
}
```

Update signature field

This request updates a signature field in a pdf document.

i The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/signaturefield/{fname}`

i *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes and produces

JSON, XML

Header

Accept: application/json, application/xml

Method

PUT

Example request

```
PUT http://localhost:6610/sdweb/rest/v5/documents/42/signaturefield/
signature-1
```

Example header

```
Accept: application/json
```

Path parameters

- **docid** (string, required): The ID of the related pdf document
- **fname** (string, required): The (form) field name

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

i The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to update a field in a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'
```

```
sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

Body input

The body contains the `RestSignatureFieldInput` structure in JSON or XML format.

Body parameters in `RestSignatureFieldInput` structure

- **captureFieldSubtypeChoice** (List<`ERestCaptureFieldSubtype`>, optional): A signature subtype can be a signature, an image or a “click to sign” (c2s) signature. The input is a comma separated list of signature subtypes which should be offered to a signer for signing a signature field.
Possible input values: `CFST_SIGNATURE`, `CFST_C2SSIGNATURE`, `CFST_IMAGECAPTURE`
- **id** (string, optional): The ID of the field in SignDoc.
- **name** (string, optional): The name of the field, must be unique within the document
- **signerId** (string, optional): If the signature should be captured by a specific person, the ID of this person can be set here. The value is stored as metadata in the ‘encrypted’ collection of the SignDoc document properties with the key “`SIGNDOCWEB_FIELD_SIGNER_`” + (field) name.
Example: `SIGNDOCWEB_FIELD_SIGNER_sig1`
- **signerName** (string, optional): If the signature should be captured by a specific person, the name of this person can be set here. The value is stored as metadata in the ‘encrypted’ collection of the SignDoc document properties with the key “`SIGNDOCWEB_FIELD_SIGNER_`” + (field) name.
Example: `SIGNDOCWEB_FIELD_SIGNER_sig1`
- **required** (boolean, optional): Defines whether the new signature field is a required field in the document which must be signed or if capturing of a signature is optional. Default is optional. Allowed values are true or false.
Default value: false
Note: The flags `required` and `readOnly` cannot be set to true for the same field.
- **readOnly** (boolean, optional): Defines whether the new signature field is a read only field in the document which cannot be signed. Default is signable. Allowed values are true or false.
Default value: false
Note: The flags `required` and `readOnly` cannot be set to true for the same field.
- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **tooltip** (string, optional): The tooltip of the field could be also displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the ‘encrypted’ collection of the SignDoc document properties with the key “`SIGNDOCWEB_TOOLTIP_`” + (field) name.
Example: `SIGNDOCWEB_TOOLTIP_sig1`
- **widgets** (List of `RestWidget` objects, required): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible “widgets”. For instance, a radio button group (radio button field) usually has multiple visible buttons, i.e, widgets.

RestWidget structure

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).

- **pageNumber** (integer, optional): The page number within the document.
- **top** (double, optional): Set the top coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **left** (double, optional): Set the left coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **right** (double, optional): Set the right coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **bottom** (double, optional): Set the bottom coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).

Example request

```
PUT http://localhost:6610/sdweb/rest/v5/documents/42/signaturefield/
signature-1?packageid=package1&fid=a2348a11-294b-4336-c11a-aaa34590887d
```

Example request body (JSON)

```
{
  {
    "restSignatureFieldInput": {
      "captureFieldSubtypeChoice": ["CFST_SIGNATURE"],
      "required": true,
    }
  }
}
```

Response status

Status code 200 (OK): The field could be successfully updated.

Status code 404 (NOT FOUND): The document was not available in the requested session.

In case of status code 200 the response body contains the complete definition of the updated signature field in a `RestDocumentOutput` structure

Examples of response body


```
{
  "restDocumentOutput" : {
    "id" : "signdocbdo - 022fda5b-acc3-8825a-8348-a3e456afd9c4",
    "signatureFields" : [ {
      "captureFieldSubtypeChoice": ["CFST_SIGNATURE"],
      "name" : "signature-1",
      "required" : true,
      "readOnly" : false,
      "alternateName" : "Signature field 1",
      "tooltip" : "This is signature field number 1",
      "widgets" : [ {
        "index" : 0,
        "pageNumber" : 1,
        "left": 511.12,
        "top": 120.02,
        "right": 591.12,
        "bottom": 80.12
      } ]
    } ]
  }
}
```

Clear signature field

This request returns a snapshot of a previously loaded document which must be available in the same session (see [Preload PDF document with commands and prepare options](#) and [Activate preloaded PDF document for processing](#)).

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}/signaturefields/{fname}/signature`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Cookie: JSESSIONID=...

Method

DELETE

Example request

```
DELETE http://localhost:6610/sdweb/rest/v5/documents/203932-12342-fdbdfg/
signaturefields/2r3-h546h63-234523/signature
```

Path parameters

- **docId** (string, required): Document ID (which was returned from a document preload request with `init=true`)
- **fname** (string, required): Signature field name from the PDF document

Query parameters


- **fid** (string, optional): The field ID in SignDoc. This value is passed as parameter to the configured plugin.
- **fname** (string, optional): The ID of the signing package in SignDoc that is passed as parameter to the configured plugin.

Response status

Status code 200 (OK): The signature field was successfully cleared.


Insert text field

This request inserts a text field into a pdf document.

 The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/textfield`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes and produces

JSON, XML

Header

Accept: application/json, application/xml

Method

POST

Example request

POST `http://localhost:6610/sdweb/rest/v5/documents/0815/textfield`

Example header


Accept: application/json

Path parameters

- **docid** (string, required): The ID of the related pdf document

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

 The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to insert a field in a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'

sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

Body input

The body contains the `RestTextFieldInput` structure in JSON or XML format.

The specification has to provide at least the field name and one widget definition with values for left, right, top, bottom and pageNumber.

Body parameters in `RestTextFieldInput` structure

- **id** (string, optional): The ID of the field in SignDoc.
- **name** (string, optional): The name of the field, must be unique within the document
- **value** (string, optional): The value of the text field.
- **multiLine** (boolean, optional): Defines whether the new field is a single line or a multiline text field. Default is a single line text. Allowed values are true or false. Default value: false
- **maxLength** (integer, optional): Defines the maximum length of the text field value. A maximum length of a text field must be between 1 and 1024 in size. Default value: 1024
- **signerId** (string, optional): If the text value should be entered by a specific person, the ID of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDocWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDocWEB_FIELD_SIGNER_text1
- **signerName** (string, optional): If the text value should be entered by a specific person, the name of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDocWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDocWEB_FIELD_SIGNER_text1
- **required** (boolean, optional): Defines whether the new field is a required field in the document which must be filled out or if entering a text is optional. Default is optional. Allowed values are true or false.
Default value: false
Note: The flags required and readOnly cannot be set to true for the same field.
- **readOnly** (boolean, optional): Defines whether the new field is a read only field in the document which cannot be changed or if entering and changing a text value is allowed. Default is writable. Allowed values are true or false.
Default value: false
Note: The flags required and readOnly cannot be set to true for the same field.
- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **tooltip** (string, optional): The tooltip of the field could be also displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDocWEB_TOOLTIP_" + (field) name.
Example: SIGNDocWEB_TOOLTIP_text1
- **widgets** (List of `RestWidget` objects, required): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, i.e, widgets.

RestWidget structure

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).
- **pageNumber** (integer, required): The page number within the document.
- **top** (double, optional): Set the top coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **left** (double, required): Set the left coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **right** (double, required): Set the right coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **bottom** (double, required): Set the bottom coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).

Example request

```
POST http://localhost:6610/sdweb/rest/v5/documents/0815/fields?
packageid=package1&fid=f5858b96-294e-46f6-a2f4-bb6d5e90886e
```

Example request body (JSON)

```
{
  "restTextFieldInput": {
    "name": "text-1",
    "required": false,
    "maxLength": 1024,
    "alternateName": "Text field 1",
    "tooltip": "This is text field number 1",
    "widgets": [
      {
        "index": 0,
        "pageNumber": 1,
        "left": 332.25,
        "top": 489.52,
        "right": 557.25,
        "bottom": 467.02
      }
    ]
  }
}
```

Response status

Status code 201 (OK): The field could be successfully inserted.

Status code 404 (NOT FOUND): The document was not available in the requested session.

In case of status code 201 the response body contains the complete definition of the inserted text field in a RestDocumentOutput structure.

Examples of response body


```
{
  "restDocumentOutput" : {
    "id" : "signdocbdo_-_057f085c-b124-420b-89b0-18ee56afd9f0",
    "textFields" : [ {
```



```
    "name" : "text-1",
    "required" : false,
    "readOnly" : false,
    "alternateName" : "Text field 1",
    "tooltip" : "This is text field number 1",
    "widgets" : [ {
      "index" : 0,
      "pageNumber" : 1,
      "top" : 489.52,
      "left" : 332.25,
      "right" : 557.25,
      "bottom" : 467.02
    } ],
    "multiLine" : false,
    "maxLength" : 1024
  } ]
}
```


Update text field

This request updates a text field in a pdf document.

 The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/textfield/{fname}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes and produces

JSON, XML

Header

Accept: application/json, application/xml

Method

PUT

Example request

PUT `http://localhost:6610/sdweb/rest/v5/documents/0815/textfield/text-1`

Example header

Accept: application/json


Path parameters

- **docid** (string, required): The ID of the related pdf document

- **fname** (string, required): The (form) field name

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

 The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to update a field in a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'

sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

Body input

The body contains the `RestTextFieldInput` structure in JSON or XML format.

Body parameters in `RestTextFieldInput` structure

- **id** (string, optional): The ID of the field in SignDoc.
- **name** (string, optional): The name of the field cannot be changed.
Note: If provided in the update request, the name of the field is ignored.
- **value** (string, optional): The value of the text field.
- **multiLine** (boolean, optional): Defines whether the new field is a single line or a multiline text field. Default is a single line text. Allowed values are true or false. Default value: false
- **maxLength** (integer, optional): Defines the maximum length of the text field value. A maximum length of a text field must be between 1 and 1024 in size. Default value: 1024
- **signerId** (string, optional): If the text value should be entered by a specific person, the ID of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDOCWEB_FIELD_SIGNER_text1
- **signerName** (string, optional): If the text value should be entered by a specific person, the name of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDOCWEB_FIELD_SIGNER_text1
- **required** (boolean, optional): Defines whether the new field is a required field in the document which must be filled out or if entering a text is optional. Default is optional. Allowed values are true or false.
Note: The flags required and readOnly cannot be set to true for the same field.
- **readOnly** (boolean, optional): Defines whether the new field is a read only field in the document which cannot be changed or if entering and changing a text value is allowed. Default is writable. Allowed values are true or false.
Note: The flags required and readOnly cannot be set to true for the same field.

- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **tooltip** (string, optional): The tooltip of the field could be also displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_TOOLTIP_" + (field) name.
Example: SIGNDOCWEB_TOOLTIP_text1
- **widgets** (list of RestWidget objects, optional): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, ie, widgets.

RestWidget structure

- **bottom** (double, optional): Set the bottom coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).
- **left** (double, optional): Set the left coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **pageNumber** (integer, optional): The page number within the document.
- **right** (double, optional): Set the right coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **top** (double, optional): Set the top coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).

Example request

```
PUT http://localhost:6610/sdweb/rest/v5/documents/0815/textfield/text-1?
packageid=package1&fid=f5858b96-294e-46f6-a2f4-bb6d5e90886e
```

Example request body (JSON)

```
{
  "restTextFieldInput": {
    "value": "text value"
  }
}
```

Response

Status code 200 (OK): The field could be successfully updated.

Status code 404 (NOT FOUND): The document was not available in the requested session.

In case of status code 200 the response body contains the complete definition of the updated text field in a RestDocumentOutput structure.


Examples of response body

```
{
  "restDocumentOutput" : {
    "id" : "signdocbdo_-_057f085c-b124-420b-89b0-18ee56afd9f0",
```

```
    "textFields" : [ {  
      "name" : "text-1",  
      "required" : false,  
      "readOnly" : false,  
      "alternateName" : "Text field 1",  
      "tooltip" : "This is text field number 1",  
      "widgets" : [ {  
        "index" : 0,  
        "pageNumber" : 1,  
        "top" : 489.52,  
        "left" : 332.25,  
        "right" : 557.25,  
        "bottom" : 467.02  
      } ],  
      "value" : "text value",  
      "multiLine" : false,  
      "maxLength" : 1024  
    } ]  
  }  
}
```


Insert checkbox field

This request inserts a checkbox field into a pdf document.

 The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/checkboxfield`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes and produces

JSON, XML

Header

Accept: application/json, application/xml

Method

POST

Example request

POST `http://localhost:6610/sdweb/rest/v5/documents/4711/checkboxfield`

Example header


Accept: application/json

Path parameters

- **docid** (string, required): The ID of the related pdf document

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

 The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to insert a field in a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'

sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

Body input

The body contains the `RestCheckboxFieldInput` structure in JSON or XML format.

The specification has to provide at least the field name and one widget definition with values for left, right, top, bottom and pageNumber.

Body parameters in RestCheckboxFieldInput structure

- **id** (string, optional): The ID of the field in SignDoc.
- **name** (string, optional): The name of the field, must be unique within the document
- **signerId** (string, optional): If the checkbox state should be changed by a specific person, the ID of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDOCWEB_FIELD_SIGNER_cbox1
- **signerName** (string, optional): If the checkbox state should be changed by a specific person, the name of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name.
Example: SIGNDOCWEB_FIELD_SIGNER_cbox1
- **required** (boolean, optional): Defines whether the new checkbox is a required field in the document which must be set to 'checked' by the signer if it is optional. Default is optional. Allowed values are true or false.
Default value: false
Note: The flags required and readOnly cannot be set to true for the same field.
- **readOnly** (boolean, optional): Defines whether the new checkbox is a read only field in the document which cannot be changed or if toggling the state by clicking on the checkbox is allowed. Default is changeable. Allowed values are true or false.
Default value: false
Note: The flags required and readOnly cannot be set to true for the same field.

- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **tooltip** (string, optional): The ID of the field in SignDoc.
displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_TOOLTIP_" + (field) name.
Example: SIGNDOCWEB_TOOLTIP_cbox1
- **widgets** (List of RestWidget objects, required): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, i.e, widgets.

RestWidget structure

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox)
- **pageNumber** (integer, required): The page number within the document.
- **top** (double, required): Set the top coordinate. The origin is in the bottom left corner of the page (see chapter [Document coordinate system](#)).
- **left** (double, required): Set the left coordinate. The origin is in the bottom left corner of the page (see chapter [Document coordinate system](#)).
- **right** (double, required): Set the right coordinate. The origin is in the bottom left corner of the page (see chapter [Document coordinate system](#)).
- **bottom** (double, required): Set the bottom coordinate. The origin is in the bottom left corner of the page (see chapter [Document coordinate system](#)).
- **selected** (boolean, optional): This flag indicates whether the inserted checkbox should be 'On' (selected=true) or 'Off', which is the default state. Default value: false
- **buttonValue** (string, optional):
Set the buttonValue only if needed! For radio button fields and check box fields, each widget also has a "button value". The button value should remain constant after the field has been created (but it can be changed if needed). The field proper has a value which is either "Off" or one of the button values of its widgets.

Each widget of a radio button field or a check box field is either off or on. If all widgets of a radio button field or a check box are off, the field's value is "Off". If at least one widget is on, the field's value is that widget's "button value". As the value of a field must be different for the on and off states of the field, the button values must not be "Off". Default value: On

Example request

POST http://localhost:6610/sdweb/rest/v5/documents/4711/checkboxfield?
packageid=package1&fid=f5858b96-294e-46f6-a2f4-bb6d5e90886e

Example request body (JSON)

```
{
  "restCheckboxFieldInput": {
    "name": "checkbox-1",
    "required": false,
    "alternateName": "Checkbox field 1",
```

```
"tooltip": "This is checkbox field number 1",
"widgets": [
  {
    "index": 0,
    "pageNumber": 1,
    "left": 120.0,
    "top": 350.23,
    "right": 140.0,
    "bottom": 330.25,
    "selected": true
  }
]
```

Response

Status code 201 (OK): The field could be successfully inserted.


Status code 404 (NOT FOUND): The document was not available in the requested session.

In case of status code 201 the response body contains the complete definition of the inserted checkbox field in a RestDocumentOutput structure.

```
{
  "restDocumentOutput" : {
    "id" : "signdocbdo_-_072f085b-abc4-3220c-8540-23eb56aff9d2",
    "checkboxFields" : [{
      "name" : "checkbox-1",
      "required" : false,
      "readOnly" : false,
      "alternateName" : "Checkbox field 1",
      "tooltip" : "This is checkbox field number 1",
      "widgets" : [ {
        "index" : 0,
        "pageNumber" : 1,
        "top" : 489.52,
        "left" : 332.25,
        "right" : 557.25,
        "bottom" : 467.02,
        "selected": true
      } ]
    } ]
  }
}
```


Update checkbox field

This request updates a checkbox field in a pdf document.

 The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/checkboxfield/{fname}`

 *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Consumes and produces

JSON, XML

Header

Accept: application/json, application/xml

Method

PUT

Example request

```
PUT http://localhost:6610/sdweb/rest/v5/documents/4711/checkboxfield/checkbox-1
```

Example header


```
Accept: application/json
```

Path parameters

- **docid** (string, required): The ID of the related pdf document
- **fname** (string, required): The (form) field name

Query parameters

- **fid** (string, optional): The ID of the field in SignDoc
- **packageid** (string, optional): The ID of the related package in SignDoc

 The query parameters are only required if the referenced document should be also updated in SignDoc.

In order to update a field in a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<  
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentUpdate'  
  
sdweb.plugins.default.impl.documentupdate='CirrusDocumentUpdate'
```

Body input

The body contains the `RestCheckboxFieldInput` structure in JSON or XML format.

Body parameters in RestCheckboxFieldInput structure

- **alternateName** (string, optional): An alternate name of the field which could be used to display a more readable name for the field.
- **clickedIndex** (integer, optional): It is possible to toggle the state of the checkbox by setting the widget index in this parameter in the update request. This simulates the click on the checkbox field from 'On' to 'Off' or from 'Off' to 'On'. This parameter can be set to change the state of the checkbox instead of the selected flag in the RestWidget structure (which has higher priority).
- **id** (string, optional): The ID of the field in SignDoc.
- **name** (string, optional): The name of the field, must be unique within the document.
- **required** (boolean, optional): Defines whether the new checkbox is a required field in the document which must be set to 'checked' by the signer if it is optional. Default is optional. Allowed values are true or false.
Note: The flags required and readOnly cannot be set to true for the same field.
- **readOnly** (boolean, optional): Readonly field in the document which cannot be changed or if toggling the state by clicking on the checkbox is allowed. Default is changeable. Allowed values are true or false.
Note: The flags required and readOnly cannot be set to true for the same field.
- **signerId** (string, optional): If the checkbox state should be changed by a specific person, the ID of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name. Example: SIGNDOCWEB_FIELD_SIGNER_cbox1
- **signerName** (string, optional): If the checkbox state should be changed by a specific person, the name of this person can be set here. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_FIELD_SIGNER_" + (field) name. Example: SIGNDOCWEB_FIELD_SIGNER_cbox1
- **tooltip** (string, optional): The tooltip of the field could be also displayed as a hint or for a better understanding of the purpose of the field. The value is stored as metadata in the 'encrypted' collection of the SignDoc document properties with the key "SIGNDOCWEB_TOOLTIP_" + (field) name. Example: SIGNDOCWEB_TOOLTIP_cbox1
- **widgets** (list of RestWidget objects, optional): A widget (annotation) structure contains information about the visible part of a field like the position and size. In PDF documents, a field may have multiple visible "widgets". For instance, a radio button group (radio button field) usually has multiple visible buttons, i.e. widgets.

RestWidget structure

- **bottom** (double, optional): Set the bottom coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **buttonValue** (string, optional): Set the buttonValue only if needed! For radio button fields and check box fields, each widget also has a "button value". The button value should remain constant after the field has been created (but it can be changed if needed). The field proper has a value which is either "Off" or one of the button values of its widgets. Each widget of a radio button field or a check box field is either off or on. If all widgets of a radio button field or a check box are off, the field's value is "Off". If at least one widget is on, the field's value is that widget's "button value". As the value of a field must be different for the on and off states of the field, the button values must not be "Off".

- **index** (integer, optional): The 0-based index number of the widget. The index number is needed if the field contains more than one widget, e.g. in case of a radio button (group). Default is 0 in case of a field with one widget (e.g. for a signature field and usually a text field or a checkbox).
- **left** (double, optional): Set the left coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **pageNumber** (integer, optional): The page number within the document.
- **right** (double, optional): Set the right coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).
- **selected** (boolean, optional): This flag indicates whether the updated checkbox should be 'On' (selected=true) or 'Off', which is the default state. Alternatively you can set the clickedIndex to toggle the state of a checkbox.
- **top** (double, optional): Set the top coordinate. The origin is in the bottom left corner of the page (see [Document coordinate system](#)).

Example request

```
PUT http://localhost:6610/sdweb/rest/v5/documents/4711/checkboxfield/checkbox-1?packageid=package1&fid=f5858b96-294e-46f6-a2f4-bb6d5e90886e
```

Example request body (JSON)

```
{
  "restCheckboxFieldInput": {
    "widgets": [
      {
        "index": 0,
        "selected": true
      }
    ]
  }
}
```

Response status

Status 200 (OK): The checkbox field could be successfully updated. In case of status code 200 the response body contains the complete definition of the updated checkbox field in a RestDocumentOutput structure.

Status 404 (NOT FOUND): The document was not available in the requested session.

Examples of response body

```
{
  "restDocumentOutput" : {
    "id" : "signdocbdo_-072f085b-abc4-3220c-8540-23eb56aff9d2",
    "checkboxFields" : [ {
      "name" : "checkbox-1",
      "required" : false,
      "readOnly" : false,
      "alternateName" : "Checkbox field 1",
      "tooltip" : "This is checkbox field number 1",
      "widgets" : [ {
        "index" : 0,
        "pageNumber" : 1,
        "top" : 489.52,
        "left" : 332.25,
        "right" : 557.25,
```

```
        "bottom" : 467.02,  
        "selected": true  
      } ]  
    }  
  }
```

Delete document field

This request deletes a field of a pdf document.

i The referenced pdf document must be available in the requested session. The document was previously uploaded into the same session.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docid}/fields/{fname}`

i *host_server* is the host domain name or IP address, and *port_number* is the host port number (if applicable).

Produces

JSON, XML

Header

Accept: application/json, application/xml

Method

DELETE

Example request

DELETE `http://localhost:6610/sdweb/rest/v5/documents/4711/fields/field1`

Example header

Accept: application/json

Path parameters

- **docid** (string, required): The ID of the related pdf
- **fname** (string, required): The name of the field within the referenced document

Query parameters

- **fid** (string, required): The ID of the field in SignDoc
- **packageid** (string, required): The name of the field within the referenced document

i The query parameters are only required if the referenced document is loaded from SignDoc.

In order to delete a field from a SignDoc document the request must contain a valid X-AUTH-TOKEN header for authentication.

The following entries must be added in `sdweb_config.groovy` (for usage with SignDoc):

```
sdweb.plugins.loadlist <<
'de.softpro.sdweb.plugins.impl.cirrus.CirrusDocumentFieldDelete'

sdweb.plugins.default.impl.fielddelete='CirrusDocumentFieldDelete'
```

Example request

```
DELETE http://localhost:6610/sdweb/rest/v5/documents/4711/fields/field1?
packageid=pac1&fid=7e6d2be9-b5f7-466a-be62-97955cdc5b25
```

Response status

Status 200 (OK): The field could be successfully deleted.

Status 404 (NOT FOUND): The document was not available in the requested session or the field does not exist in the referenced document.

Archive document

This request archives the current session document (see [Preload PDF Document with Commands and Prepare Options](#) and [Activate Preloaded PDF Document for Processing](#)). The document is archived via preconfigured and preloaded DMS plugin (see [Plugin interface](#)).

It is necessary to provide the DMS ID for addressing a specific DMS plugin. This DMS ID can be specified as value for `dmsid` either during upload and prepare of the document (see [Preload PDF Document with Commands and Prepare Options](#)) or with parameter `dmsid` (with higher priority) directly in the archive document call.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}/archive`

Produces

JSON, XML

Header

Accept: application/json, application/xml

Cookie: JSESSIONID=...

Method

POST

Example request

```
POST http://localhost:6610/sdweb/rest/v5/documents/203932-12342-fdbdfg/
archive?dmsid=de.softpro.sdweb.plugins.impl.FileDms
```

Path parameters

- **docId** (string, required): Document ID (which was returned from a document preload request with init=true)


The request body must hold a checkbox field object with the following structure.

Query parameters

- **dmsid** (string, optional): The ID of the DMS plugin that should archive the document. The DMS plugin must be configured and loaded in the SignDoc Web server before it can be used.

Response status

Status is 200 (OK): The document could be successfully archived.

 The Validator plugin (with method validateDocumentBeforeArchive) which can be defined via validatepluginid in 'Upload document' request is not called with 'Archive document' request.

Remove document

This request removes the previously loaded document from session workspace.

URL

`http://host_server:port_number/sdweb/rest/v5/documents/{docId}`

Produces

JSON, XML

Header

Accept: application/json, application/xml

Cookie: JSESSIONID=...

Method

DELETE

Example request

```
DELETE http://localhost:6610/sdweb/rest/v5/documents/123
```

Path parameters

- **docId** (string, required): Document ID (which was returned from a document preload request with init=true)

Response status

Status is 200 (OK): The requested document could be successfully removed from the session workspace. Otherwise 404 (NOT FOUND), if ID not found or invalid. A SignDoc Web status code is then returned together with an explaining message.

Document coordinate system

The origin of the document coordinate system is in the bottom left corner of the page (as rendered, that is, taking rotation of PDF pages into account). Points having positive X coordinates are to the right of the origin, points having positive Y coordinates are above the origin.

For PDF documents, the origin is in that corner of the intersection of the CropBox and the MediaBox of the page that corresponds to the bottom left corner of the image that would be rendered for that page. The units are specified by the PDF document and are usually 1/72 inch.

For TIFF documents, the origin is in the bottom left corner of the page, the unit is one pixel.

Chapter 5

Plugin interface


For plugin interface description see separate documentation shipped with SignDoc Web.

Documentation:

`SDWEB_HOME/interfaces/plugins/sdweb-plugins-<VERSION_NUMBER>-javadoc.zip`

Samples with sources:


`SDWEB_HOME/interfaces/plugins/sdweb-samples-plugins-<VERSION_NUMBER>-sources.zip`

 Plugins developed for previous SignDoc Web versions can usually be used with SignDoc Web 5.1. To enable this the SignDoc Web administrator has to rename or copy the plugin directory to `SDWEB_HOME/plugins/V3/default`.

SignDoc Web plugins general information

Description

SignDoc Web provides many plugin interfaces enabling the customer to adapt the server to their own needs. The plugin interfaces are available as Java interfaces.

 All SignDoc Web plugins need to implement their interface methods in a thread-safe way, since each plugin is instantiated only once by the server.

Plugin development

The Java development package is available in the directory:

`SDWEB_HOME/interfaces/plugins`

When developing a plugin, all of the files of this directory should be added to the build class path. The relevant files are:

- `sdweb-plugins-VERSION_NUMBER.jar`

Example

`sdweb-plugins-2.1.0.0.0.34.jar`

This is the main jar file that contains all plugin interfaces.

- `sdweb-plugins-VERSION_NUMBER-javadoc.zip`

Example

sdweb-plugins-2.1.0.0.0.34-javadoc.zip

This is the JavaDoc jar file for the corresponding sdweb-plugins-VERSION_NUMBER.jar file.

- sppluguin-fw-VERSION_NUMBER.jar and sppluguin-if-VERSION_NUMBER.jar

Example

sppluguin-fw-2.1.0.0.0.34.jar and sppluguin-if-2.1.0.0.0.34.jar

Required dependencies for sdweb-plugins-VERSION_NUMBER.jar

General information

Each plugin must have its own plugin ID (PLUGIN_ID) which the SDWEB server uses to reference it.

Best practices

When writing your own plugin, it is recommended to derive your plugin class from the class `de.softpro.sppluginif.AbstractPlugin` and implement the required interface. This ensures, that the PLUGIN_ID (see above) equals the class name of the plugin.

Example

```
public class MyPlugin extends AbstractPlugin implements IDms {  
    ...plugin code...  
}
```

Supported plugin packaging

The SignDoc Web server supports different kinds of plugin packagings.

Simple class files (TYPE-1-PACKAGE)

This is applicable, if the plugin has no dependencies to 3rd party libraries and only uses classes of the JRE and classes available in the SignDoc Web Context's classpath.

Jar file (TYPE-2-PACKAGE)

This is basically the same as above. The only difference is that the class files are packed as a standard jar file.

Zip file (TYPE-3-PACKAGE)

This is applicable, if the plugin is more complex and has dependencies on other 3rd party libraries that are not available on the class path. If the package method shall be used, the zip file needs to have the following sub structure:

```
MyPlugin.zip  
|  
|--> /classes  
|   |  
|   |--> simple class, properties, xml files  
|  
|--> /lib  
|   |  
|   |--> jar files
```


i The plugin class path overrides the web-app classpath. This means that if a class of the plugin package references a class, and the same class exists in the server context and the plugin package. The class of the plugin package will be used.

Classpath order:

1. classes directory

This directory contains simple class files and other files like e.g. properties or other configuration files.

2. lib directory

This directory contains standard java jar files.

3. SignDoc Web Context

The standard class path of the signdoc web context as provided by the application container

Plugin installation and usage

The plugin and usage consists usually of the following steps:

1. Copy the plugin package in the SignDoc Web plugin directory.
2. Load the plugin in `sdweb_config.groovy`.
3. Verify that the plugin was loaded.
4. Use the plugin.

The plugin directory is located in:

`SDWEB_HOME/plugins/V3`

The default deployment structure looks like this:

```
SDWEB_HOME/plugins/V3/default
|
--> /classes
|   |
|   --> simple class, properties, xml files (TYPE-1-PACKAGE)
|
--> /lib
|   |
|   --> jar files (TYPE-2-PACKAGE)
|
--> myPlugin1.zip (TYPE-3-PACKAGE)
|
--> /myPlugin1 (directory will be automatically created on SignDoc Web start and
contains the contents of myPlugin1.zip)
|   |
|   --> classes
|   |
|   --> lib
|
--> myPlugin2.zip (TYPE-3-PACKAGE)
|
--> /myPlugin2 (directory will be automatically created on SignDoc Web start and
contains the contents of myPlugin2.zip)
```

```
|
|
|      --> classes
|      |
|      --> lib
|
|--> ... (more TYPE-3-PACKAGEs)
```

Step 1 - Copy the plugin package in the SignDoc Web plugin directory

If your plugin is packaged as:

- TYPE-1-PACKAGE put your class files in the directory:

`SDWEB_HOME/plugins/V3/default/classes`

- TYPE-2-PACKAGE put your jar files in the directory:

`SDWEB_HOME/plugins/V3/default/lib`

- TYPE-3-PACKAGE put your zip files in the directory:

`SDWEB_HOME/plugins/V3/default`

These zip files must conform to the schema described above (see Supported Plugin packaging TYPE-3-PACKAGE) to be used correctly.

Step 2 - Load the plugin in `sdweb_config.groovy`


You need to know the class name (`PLUGIN_CLASSNAME`) of your plugin to load it.

Open the `sdweb_config.groovy` file in an editor:

`SDWEB_HOME/conf/sdweb_config.groovy`

Search for a line starting with `sdweb.plugins.loadlist`.

If the line does not exist add a new line at the end of the `sdweb_config.groovy` file.

 The `PLUGIN_CLASSNAME` below needs to be put in quotes. If not, this will result in an configuration error and the plugins are not loaded.

```
sdweb.plugins.loadlist = ["PLUGIN_CLASSNAME"]
```

Example


```
sdweb.plugins.loadlist = ["com.example.MyPlugin"]
```

If you want to load multiple plugins you just extend the list:

```
sdweb.plugins.loadlist = ["PLUGIN_CLASSNAME_1", "PLUGIN_CLASSNAME_2", ...]
```

Example

```
sdweb.plugins.loadlist = ["com.example.MyPlugin", "com.example.AnotherPlugin"]
```

 Restart the SignDoc Web server to load the plugins.

Step 3 - Verify that the plugin was loaded

The easiest way to do this, is to open the SignDoc Web About page and look in the plugins section. If the plugin was loaded, it should appear in the list. If there is a problem, a red or yellow line will report the issue.

```
INFO *** Available Plugins...
INFO PluginID:de.softpro.sdweb.plugins.impl.InternalTimestampProvider
INFO PluginID:de.softpro.sdweb.plugins.impl.TemplateDocumentMonitor -
INFO PluginID:de.softpro.sdweb.plugins.impl.SimpleFilePreloader - Sim
INFO PluginID:de.softpro.sdweb.plugins.impl.DefaultPreparePlugin - Sig
INFO PluginID:de.softpro.sdweb.plugins.impl.SimpleC2SSVGRenderer - Sir
INFO PluginID:de.softpro.sdweb.plugins.impl.DefaultResultParams - Sig
INFO PluginID:de.softpro.sdweb.plugins.impl.FileDms - SignDocWeb File
INFO PluginID:de.softpro.sdweb.plugins.impl.SimpleAuditLog - Simple Av
INFO PluginID:de.softpro.sdweb.plugins.impl.DefaultValidator - SignDoc
INFO PluginID:de.softpro.sdweb.plugins.impl.PDF2TIFFConversion - Plug
de.softpro.sdweb.plugins.impl.PDF2TIFFConversion
INFO PluginID:de.softpro.sdweb.plugins.impl.BasicAuthenticator - Signl
```

Step 4 - Use the plugin

The plugin usage depends on the type of the plugin:

DMS-Plugin (IDms, IDmsEx1)

The plugin is specified, when opening a document as servlet parameter dmsid.

Example request for using the plugin with the ID com.example.MyPlugin:

```
<form action="http://www.signdocweb.com/sdweb/load/byurl" target="_blank" method="post"
>
  <input name="docurl" value="DOCUMENT_URL" type="hidden"/>
  <input name="dmsid" value="com.example.MyPlugin" type="hidden"/>
  <input type="submit" value="open"/>
</form>
```

Available plugin interfaces

Prepare plugin interface

This plugin is called just after the document is created and form and metadata is prefilled. The plugin can parse the provided metadata and set new metadata or populate form field by returning an IDocumentData object to the caller.

- Java interface to implement
 - Name: IPrepare
 - Package: de.softpro.sdweb.plugins.prepare
- Servlet parameter
 - Name: preparepluginid
 - Value: Plugin ID

Signature archive plugin interface

This plugin is called whenever a taken signature needs to be validated against a reference signature or a signature should be stored in the signature archive.

- Java interface to implement
Name: ISignatureArchive
Package: de.softpro.sdweb.plugins.sigarchive
- Servlet parameter
Name: signaturearchiveid
Value: Plugin ID

Document validator plugin interface

This plugin is called whenever the contents of a document needs to be validated for correctness.

- Java interface to implement
Name: IDocumentValidator
Package: de.softpro.sdweb.plugins.validate
- Servlet parameter
Name: validatepluginid
Value: Plugin ID

DMS plugin interface

This plugin is called whenever a document needs to be archived in the DMS. A document can also be loaded by this interface (optional).

- Java interface to implement
Name: IDms
Package: de.softpro.sdweb.plugins.dms
- Servlet parameter
Name: dmsid
Value: Plugin ID

IAuthenticate plugin interface

This plugin is called to authenticate the load request to the server. It may reject the request to only allow requests that are considered as valid for the implementing plugin.

- Java interface to implement
Name: IAuthenticate
Package: de.softpro.sdweb.plugins.authenticate

The used plugin is enabled in `sdweb_config.groovy` setting property:

```
sdweb.defaults.defaultauthenticator.impl
```

Example

```
sdweb.defaults.defaultauthenticator.impl=  
"de.softpro.sdweb.plugins.impl.BasicAuthenticator"
```

ISignRSA plugin interface

ISignRSA allows SignDoc Web to delegate the signature computing process to an independent process/location.

This is especially useful, if the digital signatures should be created by a CA or by using an HSM.

- Java interface to implement

Name: ISignRSA

Package: de.softpro.sdweb.plugins.signing

To use the ISignRSA interface, implement a de.softpro.sdweb.plugins.signing.ISignRSA as documented in the SignDoc Web plugin developer documentation (JavaDoc). A sample plugin with source code is also provided in the samples package.

To use it the plugin has to be

- loaded in `sdweb_config.groovy` as any other plugin
- set as signing plugin in `sdweb_config.groovy`

Setting the SignRSA plugin as signing plugin:

```
[sdweb_config.groovy]
sdweb.rsa.pluginid=<THE_PLUGIN_ID>
```

Example

```
sdweb.rsa.pluginid="de.softpro.sdweb.plugins.impl.demo.SignRSADemo"
```

Resources:

- JavaDoc is located in %SDWEB_HOME%\interfaces\plugins\sdweb_plugins-VERSION_NUMBER-javadoc.zip
- Sample code can be found in: %SDWEB_HOME%\interfaces\plugins\sdweb-samples-plugins-VERSION_NUMBER-sources.zip

General SignDoc Web plugin interface

The SignDoc Web plugin interface makes a plugin usable from within SignDoc Web.

The interface definition is located in the directory:

```
%SDWEB_HOME%/interfaces/plugins/sdweb-plugins-VERSION_NUMBER.jar
```

The documentation is located in:

```
%SDWEB_HOME%/interfaces/plugins/sdweb-plugins-VERSION_NUMBER-javadoc.zip
```

Load configuration

Which plugins should be loaded is defined in the configuration file:

```
[[sdweb_config.groovy]]
```

The following entry has to be added or modified in this configuration file:

```
loadlist = ['PLUGIN', 'PLUGIN',]
```

```
sdweb{
  plugins {
    loadlist = ['de.softpro.sdweb.plugins.impl.FileDms',
'de.softpro.sdweb.plugins.impl.SignArchive',]
  }
}
```

Each plugin that should be loaded on startup must appear in this loadlist and be in the class path of the web application. The contents of the loadlist is the class name of the plugin. The plugins

```
'de.softpro.sdweb.plugins.impl.FileDms',
'de.softpro.sdweb.plugins.impl.SignArchive'
```

will be loaded as default.

Install plugin

Plugins are loaded from the following directory:

```
%SDWEB_HOME%/plugins/V3/default
```

Deploy plugin

The Kofax recommended way to deploy a plugin is to create a ZIP file name `PLUGINNAME.zip` containing the following structure:

```
PLUGINNAME.jar
\classes ()
\lib(containing other necessary libs to run the plugin)
```

If the plugin directory contains files with extension '.zip' then the SignDoc Web plugin deploy mechanism tries to create a (sub)directory with the name of the ZIP file (without extension '.zip'). Within that directory the two sub-directories will be created:

```
/classes
```

```
/lib
```

If the `PLUGINNAME.jar` contains a package structure, that package structure will be created within the classes sub-directory and the classes will be deployed according to the package structure.

Example

A file `MyDMS.zip` containing `MyDMS.jar` would result in creating the sub-directories `MyDMS`,

```
MyDMS/classes
```

```
MyDMS/lib
```

The deploy process of `MYDMS.jar` based on the package structure `de.softpro.sdweb.plugins.impl.MYDMS` creates the following structure:

```
MYDMS.jar
\classes
  \de
    \softpro
      \sdweb
        \plugins
          \impl
```

```
\lib                                     \MYDMS.classes
```

IPlugin

All SignDoc Web plugins must implement the Java interface

```
de.softpro.sppluginif.IPlugin
```

If this interface is implemented a plugin can be loaded in SignDoc Web.

AbstractPlugin

The class

```
de.softpro.sppluginif.AbstractPlugin
```

is a recommended convenience class for implementing a plugin, since it implements already a number for functions with sensible values:

- PluginId is mapped to the class name
- PluginInterfaceVersion is mapped to the correct server version

A plugin extending this class can be loaded in SignDoc Web.

Interesting functions

Injecting plugin configuration

```
void injectPluginConfiguration(Map<String, Object> configMap);
```

This method gets called in the loading phase of a plugin and it passes a map of plugin configuration values using key/value pairs.

The configuration is done in the `sdweb_config.groovy` file using this convention:

```
sdwebplugins {  
    "<pluginid_as_defined_in_IPlugin>" {  
        key=value  
    }  
}
```

where

key: is always treated as string

value: can be any Java type like e.g. boolean, int, string

Example

```
sdwebplugins {  
    "de.softpro.sdweb.plugins.impl.FileDms" {  
        dmsFolder = "$sdWebHome/dms/de.softpro.sdweb.plugins.impl.FileDms"  
        fileLockTimeout = 10  
    }  
    "de.softpro.sdweb.plugins.impl.SignArchive" {  
        webservice {  
            url = "http://localhost:2100/services/signature"  
        }  
    }  
}
```

```
}  
}
```

The values of the above settings can be accessed in the following way:

```
"de.softpro.sdweb.plugins.impl.FileDms"
```

```
String myDmsFolder = configMap.get('dmsFolder');  
int myFileLockTimeout = configMap.get('fileLockTimeout');
```

```
"de.softpro.sdweb.plugins.impl.SignArchive"
```

```
String myWebServiceUrl = configMap.get('webservice.url');
```

Returning error messages

```
String getErrorMessage(IPluginError pluginError, Locale locale);
```

Each plugin is responsible for returning its own error messages.

Every plugin should throw an:

```
de.softpro.sppluginif.PluginException
```

that implements the `IError` interface in case of a bad error and wait for the server to query the error string from the plugin.

The parameter `locale` can be used to provide a translated error message.

Trusted service provider

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:

- **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:

- **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:

- **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.
- **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:

- **OUT_RESULT**

A boolean value indicating the result of the validation. True denotes a successful validation of the credentials for the specified signature type.

In case of processing errors, an appropriate exception will be thrown.

TSP post document signature event

The `de.softpro.cirrus.plugins.event.tsp.PostDocumentSignatureEvent` starts the signing process with the TSP provider. The input parameters are:

- **IN_CREDENTIALS**
The credentials, according to the information provided by the info event. Credentials are provided as a map with the parameter ID as a key and the value given for that parameter. To avoid exceptions, parameters should be validated with the regular expression returned by the info event.
- **IN_AUTHENTICATION_TOKEN**
An authentication token, if available.
In case the TSP uses a short lived authentication token to authenticate the service, the token can be passed here.
- **IN_SIGNATURE_TYPE**
The requested signature type (BASIC, ADVANCED or QUALIFIED).

- **IN_DOCUMENT**
The document to be signed (byte array).
- **IN_DOCUMENT_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:

- **OUT_AUTHENTICATION_TOKEN**
A new authentication token after successful authentication, in case the TSP uses an authentication token mechanism.
- **OUT_DOCUMENT_VERIFICATION_URL**
The TSP document verification URL.
The Signing Client will redirect to this TSP URL letting the signer authenticate with the TSP service and sign the document.
- **OUT_SIGNATURE_PROCESS_TOKEN**
The signature token returned by the TSP after initiating the document signing. This token is used to identify this particular signing process.

TSP get document signature event

The `de.softpro.cirrus.plugins.event.tsp.GetDocumentSignatureEvent` is used to retrieve a signed document from the TSP previously sent for signing.

The input parameters are:

- **IN_AUTHENTICATION_TOKEN**
An authentication token, if available.
In case the TSP uses a short lived authentication token to authenticate the service, the token can be passed here.
- **IN_SETTINGS**
Optional. If present, the settings map will completely override the settings provided at plugin instantiation via `IPlugin.injectPluginConfiguration`. This is necessary, because SignDoc Web has no account-specific plugin mechanism, thus the account-specific parameters need to be passed for each call. The plugin has to be able to re-initialize on a 'by call' basis.
- **IN_SIGNATURE_PROCESS_TOKEN**
The signature token that identifies this signing process (received by `PostDocumentSignatureEvent`).

The output data includes:

- **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.

Chapter 6

Script plugins

SignDoc Web can execute plugins that are written in a script language. Unlike traditional plugins that are developed with the Java plugin SDK, the script plugins do not need an SDK and do not need the Java language.

Supported script Languages [as of SignDoc Web 5.1]


- Groovy 2.2

Plugins providing script implementation [as of SignDoc Web 5.1]

- `de.softpro.sdweb.plugins.impl.ScriptIPrepareEx`
provides script implementation for these interfaces
 - `de.softpro.sdweb.plugins.prepare.IPrepareEx`

General usage

After starting the server, you will find certain script files that represent a plugin interface in `SDWEB_HOME/plugins/<version>/scripts`. To fill the interface with life just edit the files with a text editor. The files can be edited while the server is running. The semantics and documentation of the plugin functions can be found in the JavaDoc of the SignDoc Web plugin SDK.

 If a plugin method expects a return value then this is always of type `AtomicReference`.

The header of the script file already contains many useful information on how to use the plugin. To use this plugin as default plugin for every document loaded you set it as default plugin in `sdweb_config.groovy`.

```
// define this plugin as default plugin...
sdweb.plugins.global.preparepluginid.pluginid=
"de.softpro.sdweb.plugins.impl.ScriptIPrepareEx" // (written as one
line)
```

Example for SignDoc Web 5.1 / `IPrepareEx` plugin

Directory: `plugins/V3/scripts/de.softpro.sdweb.plugins.impl.ScriptIPrepareEx`

File: `getCmdStatements.groovy` (implements the functionality as described in JavaDoc of the SignDoc Web plugin SDK)

Appending the following lines to the end of the `getCmdStatements.groovy` file will insert 2 signature fields at the bottom of every loaded pdf document that uses the plugin

```
de.softpro.sdweb.plugins.impl.ScriptIPrepareEx
```

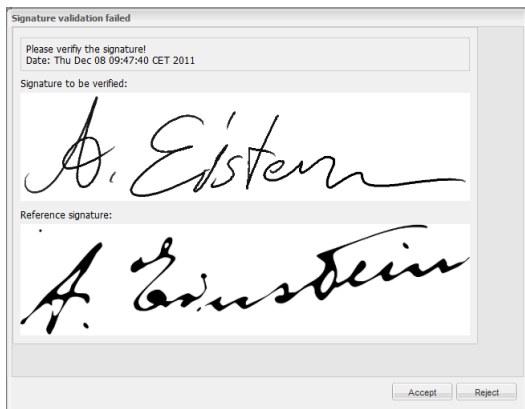
```
println "***** !!! script plugin was called !!! *****"
```

```
def command_list = [  
  'name=sig1|page=1|bottom=10|left=10|width=100|height=50|type=formfield|  
  subtype=signature',  
  'name=sig2|page=1|bottom=10|left=120|width=100|height=50|type=formfield|  
  subtype=signature'  
]  
// return value is an AtomicReference Object  
my_sdweb_plugin_return_value.set(command_list)
```

Chapter 7

Online signature verification enhancements

SignDoc Web and the plugin interface `ISignatureArchiveEx2` was extended to support a more sophisticated "Online Signature Verification" workflow.



Prerequisites

Resources for plugin developers

See [General SignDoc Web plugin interface](#)

Plugin interfaces to implement

- extend `de.softpro.sppluginif.AbstractPlugin`
- implement `de.softpro.sdweb.plugins.sigarchive.ISignatureArchiveEx2`

alternative way

- implement `de.softpro.sdweb.plugins.sigarchive.ISignatureArchiveEx2`
- implement `de.softpro.sppluginif.IPlugin`

Description

If the SignDoc Web plugin interface

`de.softpro.sdweb.plugins.sigarchive.ISignatureArchiveEx2` is implemented by a plugin, the plugin is usually used to compare a test signature against a reference signature. The result of such a compare action is `ValidationResult` object. This object can return a `VALIDATION_RESULT` state back to SignDoc Web and so control the further workflow.

i The implementing plugin is called every time, after signature is captured and before the document is signed. This allows the workflow to check a signature before actually using it.

Possible VALIDATION_RESULT values and their implication on the workflow:

VALIDATION_RESULT	Description
MATCH	The signature to validate will be accepted and the signature field is signed.
NO_MATCH	The signature to validate will be rejected and the signature field is not signed.
NOT_FOUND	The GUI will display a message, that the reference signature is missing. The signature field is not signed.
GENERIC_ERROR	The GUI will display a message, that an error occurred. The signature field is not signed
QUESTIONABLE_MATCH	The GUI will display a dialog box where the user can decide, if a signature is acceptable or not. If the signature is accepted, the document will be signed.

Possible values for ValidationResult.OPTION:

Value	Java Type	Description	Applicable with
IMAGE_TEST_SIGNATURE	byte[]	An image of the test signature	QUESTIONABLE_MATCH
IMAGE_REFERENCE_SIGNATURE	byte[]	An image of the reference signature	QUESTIONABLE_MATCH
SIMPLE_GUI_MESSAGE	string	A message, the GUI should display to the user	QUESTIONABLE_MATCH
IMAGE_MIME_TYPE	string	The mime type of the test and reference image	QUESTIONABLE_MATCH
HIDDEN_PARAMETER	string	A String value, that is not touched or evaluated by the server. If a plugin needs to transport information (e.g. the match rate) from the validateSignatureEx1(..) method to the questionableSignatureDecision(..) method, it can use this parameter	QUESTIONABLE_MATCH

Usage

To use the plugin the following conditions have to be fulfilled:

- The plugin has to be loaded by SignDoc Web on Startup
- When opening a document, the load request needs to specify:
 - the `ISignatureArchive` plugin to use
 - the `signerid` for each signature field that should use the `ISignatureArchive`

sdweb_config.groovy - Load the plugin

Syntax

```
sdweb.plugins.loadlist = ["plugin class 1", "plugin class 2", ..]
```

Example

```
sdweb.plugins.loadlist = ["de.softpro.SignArchiveTestPlugin"]
```

Servlet request parameters

Parameter name	Description	Parameter value Example
signaturearchiveid	Defines which ISignatureArchive plugin to use by specifying the plugin's pluginid	de.softpro.SignArchiveTestPlugin
cmd_[unique_number]	Defines which signature field should be checked against which signer ID before accepting the signature	name=sig1 type=signaturearchive subtype=validate value=a_user_id_the_plugin_understands

Optional configuration options sdweb_config.groovy

- **sdweb.plugins.signaturearchive.validation.signWithoutReference** (boolean): If set to false, that server will not accept a signature, when the signer has no reference signature. Default: true

Code example

```
public class TestPlugin extends AbstractPlugin implements ISignatureArchiveEx2 {

    @Override
    public ValidationResult validateSignatureEx1(byte[] signature, String signerid,
        IDocumentData documentData, Map<String, Object> params) throws PluginException {

        byte[] referenceSignaturePng;
        byte[] testSignaturePng;
        String simpleGuiMessage;
        String mimeType = "image/png";

        /* Example (not useful in production)
           that sets the result based on the signerid */
        VALIDATION_RESULT valResult;
        if (signerid.equals("MATCH")) {
            valResult = VALIDATION_RESULT.MATCH;
        } else if (signerid.equals("NO_MATCH")) {
            valResult = VALIDATION_RESULT.NO_MATCH;
        } else if (signerid.equals("NOT_FOUND")) {
            valResult = VALIDATION_RESULT.NOT_FOUND;
        } else if (signerid.equals("QUESTIONABLE_MATCH")) {
            valResult = VALIDATION_RESULT.QUESTIONABLE_MATCH;
        } else if (signerid.equals("GENERIC_ERROR")) {
            valResult = VALIDATION_RESULT.GENERIC_ERROR;
        } else {
            valResult = VALIDATION_RESULT.GENERIC_ERROR;
        }

        /*****/
        /* more code here */
    }
}
```

```
/* ***** */

/* set the GUI message (only used, when valResult==QUESTIONABLE_MATCH ) */
simpleGuiMessage = "Please verify the signature!<br>Date: " + new
Date().toString();

/* a hidden parameter, the plugin can use */
options.put(OPTION.HIDDEN_PARAMETER, signerid);
/* the image of the reference signature */
options.put(OPTION.IMAGE_REFERENCE_SIGNATURE, referenceSignaturePng);
/* the image of the test signature */
options.put(OPTION.IMAGE_TEST_SIGNATURE, testSignaturePng);
/* a message, that is displayed on the GUI if VALIDATION_RESULT is
QUESTIONABLE_MATCH or NOT_FOUND*/
options.put(OPTION.SIMPLE_GUI_MESSAGE, "Please verify the signature!<br>Date: "
+ new Date().toString());
/* a valid mime-type for the images - both images must have the same file
format*/
options.put(OPTION.IMAGE_MIME_TYPE, mimeType);

ValidationResult result = new ValidationResult(valResult, options);

return result;
}

@Override
public VALIDATION_RESULT validateSignature(byte[] signature, String signerid,
IDocumentData documentData, Map<String, Object> params) throws PluginException {
    throw new IllegalAccessException("this method should not be used");
}

@Override
public void questionableSignatureDecision(String documentId, String fieldId, String
signerId, String guiQuestion, QUESTIONABLE_SIGNATURE_DECISION decision, String
hiddenParameter) {
    log.info("questionableSignatureDecision: docId=" + documentId + " fieldId=" +
fieldId + " signerId=" + signerId + " guiQuestion=" + guiQuestion + " decision=" +
decision + "hiddenParameter=" + hiddenParameter);
}
}
```