Kofax SignDoc Preproc 3.4.7

Technical Documentation

Kofax SignDoc Preproc 3.4.7 - Technical Documentation
Saved 18-May-2016

Table of Contents

1 SignDoc Preproc

Overview

The preprocessor is used to allow custom entries to be made within SignDoc Desktop® before it is opened for signing. SignDoc Desktop® supports predefined metadata entries which can be filled along with workflow assignments which can be inserted into the document and allow processing when the PDF document is later opened in SignDoc Desktop®. The preprocessor step reduces customization costs required by the customer by allowing additional information to be placed in the document which will then be processed when loaded into SignDoc Desktop®. The alterations or amendments made to the document can be performed using JavaScript. The preprocessor loads the document and uses the specified JavaScript to evaluate and amend the current document as specified.

A few general items before we get started:

- There is no GUI for the preprocessor.
- There are no options, only the PDF file name must be specified when starting.
- SignDoc Desktop® must be installed prior to using the preprocessor.
- A valid SignDoc Desktop® license is required for using the preprocessor.

Manual overview

This manual is divided into the following sections:

- Section Starting SignDoc Preproc shows the command line call and options available
- Section Reading PDF files for SignDoc Preproc explains what commands are available by SignDoc Desktop to read PDF files
- Section Metadata for SignDoc Preproc explains which metadata is supported by SignDoc Desktop

2 Using SignDoc Preproc

2.1 Starting SignDoc Preproc

SDPreproc.exe can be started via command line input or the call configured from a separate program, i.e. SignDoc PDF Printer, batch processing file, etc.

Requirements:

- SignDoc Desktop (Version 3.1.0 or higher) must be installed.
- A valid SignDoc Desktop license (may also be a demo license) is required.
- The current user must be able to start SignDoc Desktop (**SignDoc32.exe**).
- The JavaScript file **detect.js** must be available in the SignDoc Desktop installation.

Command line entry:

- **SDPreproc.exe** starts the application with a console.
- **SDPreprocW.exe** starts the application without a console.
- The name of the PDF file to be processed is required.

  Syntax

  ```
  SDPreproc.exe [PDF_FILE]
  ```

  Please specify the fully qualified path to the desired file. If there are spaces in the path please quote the entire path to the input file.

  Examples

  ```
  SDPreproc.exe c:\mytest.pdf
  SDPreproc.exe "c:\my folder\mytest.pdf"
  ```

- **SDPreproc.exe** can be started using the command line option -startapp.

  Syntax

  ```
  SDPreproc -startapp=[APP_TO_START] [PDF_FILE]
  ```

  After SDPreproc.exe has finished, the application specified after "-startapp" will be started and the current PDF document will be passed.

  Example

  ```
  SDPreproc.exe -startapp=AcroRd32 C:\temp\myfile.pdf
  ```

  The Acrobat Reader will be started after the SDPreproc has finished with

  ```
  AcroRd32 C:\temp\myfile.pdf
  ```

  Please note that AcroRd32.exe has to be available trough the PATH environment variable.

  The default setting is:

  ```
  -startapp SignDoc32
  ```

  Example

  ```
  SDPrepproc -startapp=SignDoc32 C:\temp\mypdf.pdf
  ```

  With the setting:

  ```
  -startapp none
  ```

  no application will be started.

- **SDPreproc.exe** can also be started using the command line option -jscriptfile.

  Syntax

  ```
  SDPreproc -jscriptfile=[JAVA_SCRIPT_FILE] [PDF_FILE]
  ```

  Using this option you can prepare a document with a specific JavaScript file. The default JavaScript file is **detect.js**.

  Example

  ```
  SDPreproc.exe -jscriptfile=prepare_mytest.js c:\mytest.pdf
  ```


2.2 Files Used by SignDoc Preproc

When running **SDPreproc.exe** there are further files required for executing the JavaScript components. All JavaScript files have the file name ending '.js'.

There are basically two types of JavaScript files required:

- **Detect.js** is the file called internally.
- xxxx.js files (may be many files) will be called as specified within **detect.js**.

3 Reading PDF Files for SignDoc Preproc

To allow processing of the data from the document with JavaScript the document will first be loaded by SignDoc Shared. SignDoc Shared outputs the information from the document into a XML memory file which then may be evaluated using JavaScript and allows the user to configure the metadata desired into the DocumentCollection for further SignDoc Desktop processing.

Opening a document in JavaScript

First of all load the document:

```
var pdf = loadPDF (args[0]);
```

Class PDF

The main class PDF is created by loading a PDF document with 'loadPDF'.

Then we can get the XML representation of the PDF document:

```
var doc = pdf.getXML ({mode:'lines', text:true, rectangles:true,
   rectanglelines:true, renderwidth:2000});
```

Method getXML CONFIG

Get an XML representation of the PDF document. The return value is an e4x XML object conforming to Document.dtd. CONFIG is an object whose properties control the behavior of getXML().

Options (CONFIG) supported:

Option	Description
clip	An array containing four numbers (left X coordinate, lower Y coordinate, right X coordinate, and upper Y coordinate) of a rectangle to which getXML() should be restricted. If this property is missing, the complete page will be examined.
debug	An integer containing debugging flags. Currently the following flags are available (add the values to enable multiple flags, all output goes to stdout): 1 - Print a brief representation of the rows as they are processed for rectangle and rule detection. The lengths of foreground runs are printed in parentheses, the lengths of background runs are printed without parentheses. 2 - Print information about rectangle detection.

Option	Description
	4 - Print information about rule detection. Default: 0
debugimagepath	If this property is a non-empty string, the image used for rectangle and rule detection will be written to this pathname. Default: empty string
decimals	The number of decimals to be used for coordinates in the XML document. Default: 2
grid	A boolean enabling (if true) computation and output of the vrow, vcolumn, hrow, and hcolumn attributes of the Line element. The default value is "false".
hmaxdistx	A number specifying the maximum horizontal distance in mm for row-first neighbour computation. Default: 25
hmaxdisty	A number specifying the maximum vertical distance in mm for row-first neighbour computation. Default: 1
hrules	A number specifying the minimum length of a horizontal rule in mm. If this property is 0, computation and output of horizontal Rule elements is disabled. See also rulequotient. Default: 0
mode	A string selecting how to output Flow, Para, and Line elements: "" - No output of Flow, Para, and Line elements (except for Line elements within Rectangle elements, see rectanglelines). "lines" - Output only Line elements. "nested" - Output Flow elements containing Para elements containing Line elements. Except for Para elements having an id attribute of 0, the elements are properly nested by id attribute, that is, all Para elements having the same id are grouped under the same Flow element. "pseudo-nested" - Output Flow elements containing Para elements containing Line elements. Para elements having the same id may be contained in different Flow elements. Default: ""
neighbors	A boolean enabling (if true) computation and output of the vrow, vcolumn, hrow, and hcolumn attributes of Line elements. See also hmaxdistx, hmaxdisty, vmaxdistx, and vmaxdisty. Default: false
page	The 1-based number of the page to be processed. 0 to process all pages.

Option	Description
	Default: 0
rectangleheight	A number specifying the minimum inner height of a rectangle in mm. Default: 5
rectanglelines	A boolean enabling (if true) computation and output of Line elements within Rectangle elements. Default: false
rectangles	A boolean enabling (if true) computation and output of Rectangle elements. See also rectanglewheight, rectanglelines, and rectanglewidth. Default: false
rectanglewidth	A number specifying the minimum inner width of a rectangle in mm. Default: 8
renderwidth	The width (in pixels) of the image rendered for finding rectangles and rules. Making this value too small will cause rectangles and rules to be missed. Making this value larger will cost time. Default: 2000
rulequotient	The minimum quotient of width and height for a filled rectangle to be treated as a horizontal rule, the minimum quotient of height and width for a filled rectangle to be as a vertical rule. Default: 3
text	A boolean enabling (if true) usage of Text elements inside Line elements for text. This facilitates XML search by e4x. Default: false
vmaxdistx	A number specifying the maximum horizontal distance in mm for column-first neighbour computation. Default: 1
vmaxdisty	A number specifying the maximum vertical distance for column-first neighbour computation. Default: 12
vrules	A number specifying the minimum length of a vertical rule in mm. If this property is 0, computation and output of vertical Rule elements is disabled. Note that detection of vertical rules is expensive. See also rulequotient. Default: 0

4 Metadata for SignDoc Preproc

4.1 Metadata for SignDoc PreProc

To allow further processing of the document specific items in SignDoc Desktop the information is written into the document as metadata. Metadata is a predefined data set of Name/Value pairs and attributes which are presently supported by SignDoc Desktop. The metadata is to preset SignDoc Desktop supported functions in the document prior to loading in SignDoc Desktop. This offers the capability to make alterations without requiring customization to restricted features in SignDoc Desktop.

The convenience here is that we actually do not have to place the actual data in the document, SignDoc Preproc does that for us after we are finished analyzing the document. We just have to set the data as XML Name/Value and attributes (if required) and return it back when we are finished. SignDoc Preproc actually inserts the data into the PDF document for us.

The metadata always gets placed into the DocumentCollection which is returned once the document processing in the JS module is completed.

Features which can be specified via metadata:

- Setting Collection Data for SignDoc Desktop
- Start the Signing Process
- Export the Document After Signing Completed
- Define a Signature Field
- Print the Document

4.2 Setting Collection Data for SignDoc Desktop

The DocumentCollection specifies what data is passed back to SignDoc Preproc for inserting into the document.

The collection contains all metadata which is desired or required for processing the current document. The document collection is sent as a string in the return value from your JavaScript.

```
var lastpage = parseInt( document.Page.length() );
<DocumentCollection>
  <Document StartPage="1" EndPage={lastPage}>
    <SignDocProperties>
       {dcAutoStart}
       {dcAutoExportClose}
       {dcInfoSigFieldCount}
       {dcInfo}
    </SignDocProperties>
```



```
  </Document>
</DocumentCollection>
```

4.3 Start the Signing Process

As soon as SignDoc Desktop has been started with the document loaded the signing process will automatically be started. This is extremely convenient when documents are to be signed and the amount of time and clicks required by the user are annoying.

Flags to specify to automatically starting the signing ceremony for signature fields in the document.

If the signing dialog is canceled the signature field is skipped.

If the signing dialog is closed auto signing is stopped.

```
SIGNDOC_AUTO_START_ACTIVITY
```

Predefined values:

"true" - document will be signed in signature field layout order

"false" - do nothing

Otherwise specify a comma separated list of field names to start signing after loading.

Example

```
"signaturefield1,signaturefield2,signaturefield3"
```

What do I need to set in the return data?

```
<String>
  <Name>SIGNDOC_AUTO_START_ACTIVITY</Name>
  <Value>signaturefield1,signaturefield2,signaturefield3</Value>
</String>
```

How do I do this in JS?

First of all let's define a standard string and write the text into it:

```
var dcAutoStart = '  <String>\n';
dcAutoStart += '     <Name>SIGNDOC_AUTO_START_ACTIVITY</Name>\n';
dcAutoStart += '     <Value>signaturefield1,signaturefield2,signaturefield3</Value>\n';
dcAutoStart += '  </String>\n';
```

Now whenever we are finished processing the document we want to return all values so that SignDoc Preproc inserts them in the DocumentCollection of document for us. We just add the new string (in this case we named it dcAutoStart) to the string returned to SignDoc Preproc.

```
....
return <DocumentCollection>
   <Document StartPage="1" EndPage={lastPage}>
      <SignDocProperties>
         {dcAutoStart}
      </SignDocProperties>
   </Document>
</DocumentCollection>
```

This will give us the following content in the DocumentCollection when returned:

```
....
return <DocumentCollection>
   <Document StartPage="1" EndPage={lastPage}>
```

```
        <SignDocProperties>
            <String>
                <Name>SIGNDOC_AUTO_START_ACTIVITY</Name>
                <Value>signaturefield1,signaturefield2,signaturefield3</Value>
            </String>
        </SignDocProperties>
    </Document>
</DocumentCollection>
```

4.4 Export the Document After Signing Completed

As soon as the signing process is completed in SignDoc Desktop the document will be automatically exported to the configured location and then closed. This is convenient when documents are to be saved to a specified location and the amount of time and clicks required by the user are annoying.

Example

```
SIGNDOC_AUTO_EXPORT_AND_CLOSE = true
```

What do I need to set in the return data?

```
<String>
    <Name>SIGNDOC_AUTO_EXPORT_AND_CLOSE</Name>
    <Value>true</Value>
</String>
```

How do I do this in JS?

First of all let's define a standard string and write the text into it:

```
var dcAutoExportClose = '   <String>\n';
dcAutoExportClose += '      <Name>SIGNDOC_AUTO_EXPORT_AND_CLOSE</Name>\n';
dcAutoExportClose += '      <Value>true</Value>\n';
dcAutoExportClose += '   </String>\n';
```

Now whenever we are finished processing the document we want to return all values so that SignDoc Preproc inserts them in the DocumentCollection of the document for us. We just add the new string (in this case we named it dcAutoStart) to the string returned to SignDoc Preproc.

```
....
return <DocumentCollection>
<Document StartPage="1" EndPage={lastPage}>
    <SignDocProperties>
        {dcAutoExportClose}
    </SignDocProperties>
</Document>
</DocumentCollection>
```

This will give us the following content in the DocumentCollection when returned:

```
....
return <DocumentCollection>
<Document StartPage="1" EndPage={lastPage}>
    <SignDocProperties>
        <String>
            <Name>SIGNDOC_AUTO_EXPORT_AND_CLOSE</Name>
            <Value>true</Value>
        </String>
    </SignDocProperties>
</Document>
```



```
</DocumentCollection>
```

4.5 Print the Document

Once all signatures have been completed the document can be automatically printed. This is convenient when documents are signed and the amount of time and clicks required by the user are annoying. You can specify which pages are to be printed for whom. You may also define multiple jobs to be performed on the same document.

Example

```
SIGNDOC_PRINTJOBS = ...print job ...
```

Setting	Description
name	Name of the print job. "all", "agent" and "customer" have predefined icons.
pages	Comma separated list of integers. If no pages are defined all pages will be printed out.
title	Title of the print job. The text will be displayed on the print job push button.
gui	Graphical user interface. "dialog" - The defined print job will be displayed in SignDoc Desktop with the title set. "quickprint" - Gives the same result as pressing the Print icon in SignDoc Desktop.

What do I need to set in the return data?

```
<print_jobs>
  <print_job name="agent" pages="1,2,3,6" gui="dialog">
    <title>Printout for Agent</title>
  </print_job>
  <print_job name="customer" pages="7" gui="dialog">
    <title>Printout for Customer</title>
  </print_job>
  <print_job name="all" gui="dialog">
    <title>Printout all pages</title>
  </print_job>
</print_jobs>
```

You can specify it directly in the DocumentCollection or as a variable in the document processing and the just add it to the DocumentCollection returned. How do I do this in JS?

First of all let's define a standard string and write the text into it:

```
var dcPrintJob = ' <String>\n';
dcPrintJob += '      <Name>SIGNDOC_PRINTJOBS</Name>\n';
dcPrintJob += '      <Value>\n';
dcPrintJob += '        <print_jobs>\n';
dcPrintJob += '          <print_job name="agent" pages="1,2,3,6" gui="dialog">\n';
dcPrintJob += '            <title>Printout for Agent</title>></Value>\n';
dcPrintJob += '        </print_jobs>\n';
dcPrintJob += '      </Value>\n';
dcPrintJob += '    </String>\n';
```

Now whenever we are finished processing the document we want to return all values so that SignDoc Preproc inserts them in the DocumentCollection of document for us. We just add the new string (in this case we named it dcAutoStart) to the string returned to SignDoc Preproc.

```
return <DocumentCollection>
  <Document StartPage="1" EndPage={lastPage}>
    <SignDocProperties>
      {dcPrintJob}
    </SignDocProperties>
  </Document>
</DocumentCollection>
```

This will give us the following content in the DocumentCollection when returned:

```
....
return <DocumentCollection>
   <Document StartPage="1" EndPage={lastPage}>
      <SignDocProperties>
         <String>
            <Name>SIGNDOC_PRINTJOBS</Name>
            <Value>
               <print_jobs>
                  <print_job name="agent" pages="1,2,3,6" gui="dialog">
                     <title>Printout for Agent</title>
                  </print_job>
               </print_jobs>
            </Value>
         </String>
      </SignDocProperties>
   </Document>
</DocumentCollection>
```

If you would like to have the complete document printed then:

```
....
return <DocumentCollection>
   <Document StartPage="1" EndPage={lastPage}>
      <SignDocProperties>
         <String>
            <Name>SIGNDOC_PRINTJOBS</Name>
            <Value>
               <print_jobs>
                  <print_job name="all" gui="dialog">
                     <title>Printout all pages</title>
                  </print_job>
               </print_jobs>
            </Value>
         </String>
      </SignDocProperties>
   </Document>
</DocumentCollection>
```

If you would like to have the different pages for the customer as required for the agent:

```
....
return <DocumentCollection>
   <Document StartPage="1" EndPage={lastPage}>
      <SignDocProperties>
         <String>
            <Name>SIGNDOC_PRINTJOBS</Name>
            <Value>
               <print_jobs>
                  <print_job name="agent" pages="1,2,3,6" gui="dialog">
                     <title>Printout for Agent</title>
                  </print_job>
                  <print_job name="customer" pages="7" gui="dialog">
                     <title>Printout for Customer</title>
                  </print_job>
               </print_jobs>
```



```
            </Value>
          </String>
        </SignDocProperties>
      </Document>
    </DocumentCollection>
```

4.6 Defining a Signature Field

When the PDF document is preprocessed you also have the capability to specify which signature fields you would like to have created with the respective properties.

Values:

- SIGNDOC_ADD_FIELDS_COUNT
- SIGNDOC_ADD_FIELD_*
- SIGNDOC_SIGNERPROFILE_*
- SIGNDOC_FIELDFIXED_*
- SIGNDOC_SIGNINGCEREMONY_*

SIGNDOC_ADD_FIELDS_COUNT

The total number of signature fields which are specified within the document collection must be specified.

```
<String>
  <Name>SIGNDOC_ADD_FIELDS_COUNT</Name>
  <Value>2</Value>
</String>
```

SIGNDOC_ADD_FIELD_*

This is used to add a new signature field. The '*' must be replaced with a unique number signifying the number of the file, i.e. SIGNDOC_ADD_FIELD_0, SIGNDOC_ADD_FIELD_1, ...

A list of attributes are also required for each signature field describing the name, type, location, etc. of the current field.

The attributes are:

Attribute	Description
name	The name of the signature field. No blanks are allowed. The name entered here will be displayed in the list of digital signature fields. This name will also be appended to other metadata info to signify which field is be set. **Example** `name="My_signature"`
value	The preset value when type is set to "text" or "checkbox". NOT SUPPORTED This attribute is not required for type ="signature"
type	The type of the signature field. This should always be set to 'signature'. **Example** `type="signature"`

Attribute	Description
	`type="text" NOT SUPPORTED` `type="check_box" NOT SUPPORTED`
page	The page on which the signature field is to be created. Page 1 is the first page. **Example** `page="2"`
left	The left border where the signature field is to be created. Value type: float **Example** `left="34.22"`
bottom	The lower border where the signature field is to be created. Value type: float **Example** `bottom="154.56"`
right	The right border where the signature field is to be created. Value type: float **Example** `right="123.25"`
top	The upper border where the signature field is to be created. Value type: float **Example** `top="201.01"`
signer_name	The name of the signer. If SignDoc Desktop is configured to show the signer name in the signature field, then the signer name can be set with this attribute. **Example** `signer_name="Staff"`

Example

```
<add_field name="Cashier_Mark" type="signature" page="1" left={sline1_x0}
bottom={sline1_y0} right={sline1_x1} top={sline1_y0+50} required="true"
signer_name="Staff">
</add_field>
```

SIGNDOC_SIGNERPROFILE_*

This specified the profile of the person to sign this field. The name of the signature field should be appended to the identifier. For example if the name of the field added was set to 'My_signature' then we need to specify the profile for 'SIGNDOC_SIGNERPROFILE_My_signature'.

Presently only the following profiles are supported:

 CUSTOMER

 AGENT

Example

```
<String>
  <Name>SIGNDOC_SIGNERPROFILE_My_signature</Name>
  <Value>CUSTOMER</Value>
</String>
```

SIGNDOC_FIELDFIXED_*

This allows you to set the specified signature field as fixed , not movable. The name of the signature field should be appended to the identifier. For example if the name of the field added was set to 'My_signature' then we need to specify the profile for 'SIGNDOC_FIELDFIXED_My_signature'.

Only the following values are supported:

true

false (default)

Example

```
<String>
  <Name>SIGNDOC_FIELDFIXED_My_signature</Name>
  <Value>true</Value>
</String>
```

SIGNDOC_SIGNINGCEREMONY_*

The signing ceremony defines the appearance and workflow used when requesting a signature on the tablet. The signing ceremony is optional, if not specified the standard SignDoc Desktop panel will appear on the tablet. With the signing ceremony you can select a workflow to be used when signing the document and the template(s) for the tablet display. The name of the signature field should be appended to the identifier. For example if the name of the field added was set to 'My_signature' then we need to specify the profile for 'SIGNDOC_SIGNINGCEREMONY_My_signature'.

The attributes are:

Attribute	Description
workflow	The name of the workflow desired. The workflow must be present in SignDoc Desktop, if not customization is required. **Example** `workflow="signdocextended01"` Supported values: signdocextended01 offers a single signature capture dialog signdocextended02 offers a confirmation dialog and signature capture dialog signdocextended03 offers a three and four page dialog **Example**

Attribute	Description
	`<custom_signing_ceremony workflow="signdocextended02" template1="first page.xml" template2="SignerRoleName.xml">`
template	The template describes the name of the XML template used which is always located in the config folder of SignDoc Desktop. **Example** `template1="SignerRoleName.xml"`

Additional values for variables used in the template. In the sample below 'signer_role' and 'signer_name' are required in the template **SignerRoleName.xml**:

```
<String>
   <Name>SIGNDOC_SIGNINGCEREMONY_MySignature</Name>
   <Value>
      <custom_signing_ceremony workflow="signdocextended01" template1="SignerRoleName.xml">
         <signer_role>{sOwner1_Role}</signer_role>
         <signer_name>{sOwner1Name}</signer_name>
      </custom_signing_ceremony>
   </Value>
</String>
```

Template **SignerRoleName.xml**: See SignWare background objects for more information on templates.

```
...
    <SPSWText ALignment="3" DrawFlags="2" Caption="$signer_name$">
        <SPSWCoordinate Origin="Tablet" Left="50" Right="1000" Top="50" Bottom="1000"/>
        <SPSWFont Face="Arial" Size="30"/>
    </SPSWText>
    <SPSWText ALignment="3" DrawFlags="2" Caption="$signer_role$">
        <SPSWCoordinate Origin="Tablet" Left="50" Right="1000" Top="150" Bottom="1000"/>
        <SPSWFont Face="Arial" Size="20"/>
    </SPSWText>
...
```

Sample DocumentCollection section:

```
<String>
  <Name>SIGNDOC_ADD_FIELDS_COUNT</Name>
  <Value>2</Value>
</String>
<String>
   <Name>SIGNDOC_ADD_FIELD_0</Name>
   <Value><add_field name="Cashier_Mark" type="signature" page="1" left={sline1_x0}
bottom={sline1_y0} right={sline1_x1} top={sline1_y0+50} required="true"
signer_name="Staff"></add_field></Value>
</String>
<String>
   <Name>SIGNDOC_ADD_FIELD_1</Name>
   <Value><add_field name="Signature_Customer" type="signature" page="1"
left={sline2_x0} bottom={sline2_y0} right={sline2_x1} top={sline2_y0+50}
required="true" signer_name={name}></add_field></Value>
</String>
<String>
  <Name>SIGNDOC_SIGNERPROFILE_Cashier_Mark</Name>
  <Value>AGENT</Value>
</String>
<String>
  <Name>SIGNDOC_FIELDFIXED_Cashier_Mark</Name>
  <Value>true</Value>
</String>
<String>
```



```
  <Name>SIGNDOC_SIGNERPROFILE_Signature_Customer</Name>
  <Value>CUSTOMER</Value>
</String>
<String>
  <Name>SIGNDOC_FIELDFIXED_Signature_Customer</Name>
  <Value>true</Value>
</String>
<String>
  <Name>SIGNDOC_SIGNINGCEREMONY_Signature_Customer</Name>
  <Value>
    <custom_signing_ceremony workflow="customized01"
template1="customized_cashpayment_customer1.xml"
template2="customized_cashpayment_customer2.xml">
     <zweigstelle>{zwst}</zweigstelle>
     <customer_name>{name}</customer_name>
     <betrag>{betrag}</betrag>
     <datum>{datum}</datum>
     <konto_nr>{kontonr}</konto_nr>
     <code>{code}</code>
     <uhrzeit>{uhrzeit}</uhrzeit>
     </custom_signing_ceremony>
  </Value>
</String>
```

4.7 Signature Capture Dialog Title

SIGNDOC_CAPTURE_DIALOG_TITLE_PREFIX

Value is the title of the dialog while signing corresponding signature field.

4.8 File Export Path

SIGNDOC_EXPORT_TARGET_FILEPATH

When using core (regular folder and sftp) exporting optionally specify the target filepath and filename.

If no filepath specified the filename is used, default filepath is defined in

```
signdoc2.ini export_location.output_dir
```

If no filename specified use only filepath, default filename (src filename).

5 Appendix

Contact Information

Trademarks

Copyright Notice

5.1 Contact Information

Our support team will be happy to assist you.

Kofax Customer Portal

If you need support with regards to purchased Signature products please contact us via

https://techsupport.kofax.com

We recommend to use Internet Explorer in case you experience problems with other browser software.

Address

You can also send your inquiry to the following address:

Kofax Deutschland AG
Wilhelmstrasse 34
71034 Boeblingen

Germany

5.2 Trademarks

- SIGNPLUS, SignDoc, SignWare, SignInfo, SignCheck, SIVAL, SignPad, SignSecure are registered trade marks of Kofax Inc. or its affiliates – all rights reserved.
- All other brand and product names can be trademarks, service marks or other intellectual property of the respective owners who reserve all related rights.

5.3 Copyright Notice