
# Documentation

# signoAPI Java Native 2

Software components for Java for the display and signature of PDF documents

**Version:** 3.20
**Date:** 13.11.2018


www.signotec.de
Tel.: +49 (0) 2102 53575 10
E-mail: info@signotec.de

# Contents

# 1 Document history

| Version | Date | Person responsible | Status/note |
|---|---|---|---|
| 3.0 | 21 November 2013 | Paul Grütter | Changes to sections 11, 11.1, 11.3 ff., 11.3 Item 11.2 added |
| 3.1 | 13 December 2013 | Paul Grütter | Subsection 3.1.1 added |
| 3.2 | 4 February 2014 | Paul Grütter | Changes to sections 10, 14, 17, 20 |
| 3.3 | 24 April 2014 | Paul Grütter | Changes to sections 14, 17<br>Subsection 10.1.2 added |
| 3.4 | 13 November 2014 | Paul Grütter | Changes to sections 4, 10.4, 18<br>Subsections 3.2.1 – 3.3.3, 5.25, 11.5 added |
| 3.5 | 25 February 2015 | Paul Grütter | Changes to sections 3.2, 10.6, 11, 17 |
| 3.6 | 15 January 2016 | Markus Mensinger | Gamma Pad added<br>Java 5 removed<br>Changes to sections 3.2.1, 3.2.2, 3.2.3, 3.3, 3.5, 5.25, 5.29.1, 5.29.2, 11, 12.1, 13.1 |
| 3.7 | 31 March 2016 | Markus Mensinger | Changes to sections 3.2.1, 3.2.2, 3.2.3, 21 |
| 3.8 | 24 May 2016 | Markus Mensinger | Changes to sections 10.1.1, 10.1.2, 10.1.3, 11.3.3, 11.3.4, 12.6<br>Item 21.3 added |
| 3.9 | 6 June 2016 | Markus Mensinger | Changes to sections 3.2.2, 3.2.3, 8<br>Item 21.4 added |
| 3.10 | 14 July 2016 | Markus Mensinger | Changes to section 7 |
| 3.11 | 6 September 2016 | Markus Mensinger | Changes to sections 3.2.1, 3.2.2, 3.2.3 |
| 3.12 | 14 September 2016 | Markus Mensinger | Changes to sections 3.2.1, 3.2.2, 3.2.3 |
| 3.13 | 6 February 2017 | Markus Mensinger | Changes to section 18<br>Section 23 added |
| 3.14 | 29 March 2017 | Markus Mensinger | Changes to section 18<br>Section 24 added |
| 3.15 | 8 June 2017 | Markus Mensinger | Delta Pad added, Signcap removed<br>Changes to sections 3.1, 3.2, 11<br>Sections 12, 13 removed<br>Sections 5.30-5.31, 11.6–11.9, 23 and 24 added |
| 3.16 | 30 November 2017 | Markus Mensinger | Changes to sections 3.2.1, 3.2.2, 3.2.3, 3.3, 10.1, 10.4, 11.3, 11.6-11.7, 23, 24<br>Section 3.6 added |
| 3.17 | 11 January 2018 | Markus Mensinger | Changes to section 3.2, 17 |
| 3.18 | 23 May 2018 | Markus Mensinger | Front page updated<br>Changes to section 16 |
| 3.19 | 7 June 2018 | Markus Mensinger | Changes to section 3.3 |
| 3.20 | 13 November 2018 | Markus Mensinger | Changes to sections 13, 15, 20<br>Section 25 added |

## 2 Function overview

The signoAPI Java Native 2 contains software components for the display, signing and editing of PDF-documents. The following overview lists the components contained in signoAPI Java Native 2:

| Component | Short description |
|---|---|
| de.signotec.pdf | |
| viewer | Contains the control elements for display and navigation |
| exceptions | Exceptions that may occur when loading or processing documents |
| enums | Contains enums that can be used for method calls |
| events | Interface for handling callbacks |
| signer | Provides the signature functionality for PDFs |
| dto | Contains data objects that are required for the signing |
| enums | Contains enums that may appear in data objects |
| exceptions | Exceptions that may occur when reading or signing documents |
| utilities | Provides editing functions for PDFs |
| dto | Contains data objects that are required for editing |
| exception | Exceptions that may occur when reading or editing documents |
| sample | Sample applications to demonstrate the use of the signoAPI Java Native 2 components. |

# 3 System requirements

## 3.1 Operating systems

The signoAPI Java Native 2 components are generally functional under Windows and Linux. For more details, please read the following section 'Additional components'. The signoAPI Java Native 2 was tested under the following systems:

- Windows 7
- Windows 8
- Windows 10
- Ubuntu 8.04 10.04
- Ubuntu 10.10 64-bit
- Fedora Core 4

### Linux configuration

When using HID or WinUSB devices, libusb 1.0.16 or higher is also required, which can be downloaded free of charge from http://www.libusb.org. The authorisations for libusb may also need to be adjusted so that the signotec pads can be addressed. For this, the MODE for USB must be changed to 0666:

```
# libusb device nodes
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", MODE="0666"
```

udev must be restarted afterwards:

```
udevadm control --reload-rules
```

In Debian, this setting is located in the file `/lib/udev/rules.d/50-udev-default.rules` or `/lib64/udev/rules.d/50-udev-default.rules`. The setting may be in a different location when using other distributions.

## 3.2 Additional components

The `SignoPdfSignerSTPad` interface requires the components of the signoPAD-API for Java. These components are included in this package, but may not always be up-to-date. The current version of the signoPAD-API for Java can be downloaded free of charge in the download area under www.signotec.de. The package also includes comprehensive documentation that contains details on the supported devices and system requirements.

The dependencies for the libraries of this API are listed in the following tables:

### 3.2.1 signopdf-viewer library

| Dependency | Note |
|---|---|
| bcpkix-jdk15on.jar | - |
| bcprov-jdk15on.jar | - |
| cmykprofile.jar | Optional for better representation of CMYK images |
| itext-se.jar | - |
| jpdfnotes.jar | - |
| rhino.jar | Only if validation of the form fields is enabled |

### 3.2.2 signopdf-utilities library

| Dependency | Note |
|---|---|
| bcpkix-jdk15on.jar | - |
| bcprov-jdk15on.jar | - |
| commons-codec.jar | - |
| itext-se.jar | - |
| jpdfnotes.jar | - |
| signopdf-viewer.jar<br>signopdf-signer.jar | - |
| stpad-lib.jar | - |

### 3.2.3 signopdf-signer library

| Dependency | Note |
|---|---|
| bcpkix-jdk15on.jar | - |
| bcprov-jdk15on.jar | - |
| commons-codec.jar | - |
| itext-se.jar | - |
| jna.jar<br>jna-platform.jar | - |
| jpdfnotes.jar | - |
| jpen.jar | If a pen display is used |
| jpen.dll | If a pen display is used in Windows |
| libjpen.so | If a pen display is used in Linux |
| signopdf-viewer.jar<br>stpad-native.jar | If the `SignoPdfSignerSTPad` class is used |
| stpad-lib.jar | - |

## 3.3 JRE dependencies

The signoAPI Java Native 2 components require the Java Runtime Environment (JRE) Version 1.6 or later. Both 32 bit and 64 bit versions are supported.

**Java 7**
Java version 7u76 or later is required if the application is run in an environment in which the code signing certificate is checked. Older versions of Java do not support the signature algorithm and will reject the library as unsigned. Typical use cases in which only signed code is used include Java Web Start applications and applets.

**Java 10**
In Java 10, the required JAXB API from the java.xml.bind module is no longer automatically available and must be loaded with the `--add-modules` parameter when starting the Java VM.

```
> java --add-modules java.xml.bind
```

The table below provides the Internet addresses where Java can be downloaded free of charge.

| Version | Download address |
|---|---|
| Java 6 | http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html |
| Java 7 | http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html |
| Java 8 | http://www.oracle.com/technetwork/java/javase/downloads/java- |

| | |
|---|---|
| | archive-javase8-2177648.html |
| Java 9 | http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase9-3934878.html |
| Java 10 | http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase10-4425482.html |

All components not included as standard in the JRE are included.

## 3.4 Limitations of the demo version

When using signoAPI Java Native 2 as a demo version, a watermark that refers to the demo mode is shown on each page of a loaded document. In addition, it is not possible to load or display documents in the demo version that are protected by a password.
If a document is signed with the demo version of the signoAPI Java Native 2, a corresponding watermark is embedded behind the signature.

If a signature field with the `addSignatureField()` method is generated with the demo version of the `signopdf-utilities` library, '_signotec_demo' is attached to the field name.

## 3.5 Keystores and certificates

The supplied 'signing.ks' and 'encryption.ks' keystores and the 'encryption.cer' certificate are only included for demonstration purposes. They all have the password 'password'.

It is possible to create proprietary keystores and certificates using the Java Keytool documented at https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html.

## 3.6 Security-critical data

Passwords and private keys are considered security-critical data and must be handled especially carefully. When using this API, you should observe the following precautions in addition to common security standards in order to keep your software's security level as high as possible.

- For passwords, only use data structures that can be overwritten.
  signoAPI Java uses the `char[]` data type. Data of an unchangeable type such as `string` cannot be deliberately overwritten or deleted and may remain in RAM for a very long time under certain circumstances.
- Passwords should be deleted immediately after use.
  To minimise the time frame in which passwords can be read from RAM, they should be overwritten immediately after use. signoAPI Java offers the `KeyLoader.clearPassword(char[])` method for this purpose.
- Delete private keys immediately after use.
  For Java 8 and higher versions, keys that implement the `Destroyable` interface should be overwritten/rendered unrecognisable in memory via the `destroy()` method once they are no longer needed.

# 4 Exceptions

Various methods throw exceptions in the event of an error. The exceptions that occur are derived from three different exception classes:

| Exception | Short description |
|---|---|
| de.signotec.pdf.viewer.exceptions | |
| SignoViewerException | Is thrown if an exception is intercepted internally; the internal exception can be determined using `getCause()` |
| CouldNotSaveException | Is thrown if an error occurs when saving |
| NoPdfDocumentException | Is thrown if a document cannot be processed |
| PasswordProtectedPdfException | Is thrown if a document is password protected |
| PdfUtilityException | Is thrown if an error occurs when processing a PDF |
| de.signotec.pdf.signer.exceptions | |
| SignoSignerException | Is thrown if an exception is intercepted internally; the internal exception can be determined using `getCause()` |
| InvalidParameterException | Is thrown if an invalid parameter has been transferred |
| NoCapturingException | Is thrown if the signature process has not been started yet |
| NoSignatureFieldException | Is thrown if the specified signature field does not exist |
| SignatureTooShortException | Is thrown if the captured signature is too short |
| de.signotec.pdf.utilities.exception | |
| SignoPdfUtilitiesException | Is thrown if an exception is intercepted internally; the internal exception can be determined using `getCause()` |
| InvalidParameterException | Is thrown if an invalid parameter has been transferred |

# 5 SignoViewer class

A main component is the `de.signotec.pdf.viewer.SignoViewer` class. This is a visual control element based on a JPanel to display PDF documents. The class offers the following seven constructors:

```
// 1. create empty SignoViewer
SignoViewer viewer = new SignoViewer();

try {
    // 2. open document file
    viewer = new SignoViewer("example.pdf");

    // 3. open document file and show page 4
    viewer = new SignoViewer("example.pdf", 4);

    // 4. open document stream
    viewer = new SignoViewer(new FileInputStream("example.pdf"));

    // 5. open document stream and show page 4
    viewer = new SignoViewer(new FileInputStream("example.pdf"), 4);

    // 6. open document path
    viewer = new SignoViewer("http://www.signotec.de/example.pdf");

    // 7. open document path and show page 4
    viewer = new SignoViewer("http://www.signotec.de/example.pdf", 4);
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.1 hideToolbar method

This method determines whether the viewer toolbar is to be hidden.

```
public void hideToolbar(boolean flag)
```

| Parameter | Description |
|---|---|
| `boolean flag` | Determines whether the toolbar is to be hidden |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.hideToolbar(false);
```

## 5.2 configureToolbar method

This method controls whether buttons for a document are displayed in the toolbar and sets the logo displayed in the toolbar. All buttons and the signotec logo are displayed by default. If all buttons and the logo are deactivated, the whole toolbar is hidden.

Please note: An `IllegalArgumentException` may occur when using this method with Java 7. Set the `java.util.Arrays.useLegacyMergeSort` property to `true` to rectify the problem.

```
public void configureToolbar(long bitmask)
```

```
public void configureToolbar(long bitmask, InputStream logo)
     throws SignoViewerException

public void configureToolbar(long bitmask, String logo)

public void configureToolbar(InputStream logo)
     throws SignoViewerException

public void configureToolbar(String logo)
```

| Parameter | Description |
|---|---|
| `long bitmask` | (Optional) Individual toolbar components can be activated or deactivated using this bitmask. Each set member of the `ToolBarEnum` activates the respective toolbar components. |
| `InputStream / String logo` | (Optional) `InputStream` or pad and file name for an image that is displayed as a logo. `Null` removes the logo. |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.configureToolbar(
        ToolBarEnum.GROUPPAGE.value()
        | ToolBarEnum.GROUPZOOM.value()
        | ToolBarEnum.GROUPINFO.value(),
        "signotec.png");
```

## 5.3 disableContextMenu method

This method determines whether the context menu of the viewer is deactivated.

```
public void disableContextMenu(boolean flag)
```

| Parameter | Description |
|---|---|
| `boolean flag` | Determines whether the context menu is deactivated |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.configureToolbar(false);
```

## 5.4 configureContextMenu method

This method can be used to replace the icons of the context menu.

```
public void configureContextMenu(InputStream handToolFileName, InputStream
textToolFileName, InputStream zoomToolFileName) throws SignoViewerException

public void configureContextMenu(String handToolFileName, String
textToolFileName, String zoomToolFileName) throws SignoViewerException
```

| Parameter | Description |
|---|---|
| `InputStream / String handToolFileName` | `InputStream` or path and file name of an icon to be used for the hand tool. |

| | |
|---|---|
| `InputStream / String textToolFileName` | `InputStream` or path and file name of an icon to be used for the text tool. |
| `InputStream / String zoomToolFileName` | `InputStream` or path and file name of an icon to be used for the zoom tool. |
| Return value | Description |
| - | - |

Usage:

```
viewer.configureContextMenu("hand.png", "text.png", "zoom.png");
```

## 5.5 loadDocument method

This method loads a specified document in the SignoViewer and displays it.

```
public void loadDocument(InputStream pdfDocument) throws PdfUtilityException,
PasswordProtectedPdfException, NoPdfDocumentException, SignoViewerException

public void loadDocument(String pdfDocument) throws PdfUtilityException,
PasswordProtectedPdfException, NoPdfDocumentException, SignoViewerException

public void loadDocument(URL pdfDocument) throws PdfUtilityException,
PasswordProtectedPdfException, NoPdfDocumentException, SignoViewerException
```

| Parameter | Description |
|---|---|
| `InputStream / String / URL pdfDocument` | `InputStream`, path and file name or `URL` of a PDF document |
| Return value | Description |
| - | - |

Usage:

```
try {
    viewer.loadDocument("example.pdf");
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.6 closeDocument method

This method closes a loaded document again. This method is called up automatically by `loadDocument()` before the new document is loaded.

```
public void closeDocument()
```

| Parameter | Description |
|---|---|
| - | - |
| Return value | Description |
| - | - |

Usage:

```
viewer.closeDocument();
```

## 5.7 nextPage method

This method is used to display the page following the currently displayed page.

```
public void nextPage()
```

| Parameter | Description |
|---|---|
| - | - |

| Return value | Description |
|---|---|
| - | - |

Usage:

```
viewer.nextPage();
```

## 5.8 previousPage method

This method is used to display the page preceding the currently displayed page.

```
public void previousPage()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.previousPage();
```

## 5.9 lastPage method

This method displays the last page of the loaded document.

```
public void lastPage()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.lastPage();
```

## 5.10 firstPage method

This method displays the first page of the loaded document.

```
public void firstPage()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |

| | |
|---|---|
| - | - |

Usage:

```
viewer.firstPage();
```

## 5.11 setCurrentPage method

This method switches to the specified page in the displayed document.

```
public void setCurrentPage(int page)
```

| Parameter | Description |
|---|---|
| `int page` | Number of the page to be displayed (starting at 1) |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.setCurrentPage(7);
```

## 5.12 getCurrentPage method

This method delivers the number of the currently visible page.

```
public int getCurrentPage()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `int` | The number of the currently visible page (starting at 1) |

Usage:

```
System.out.println("Current page: " + viewer.getCurrentPage());
```

## 5.13 getPageCount method

This method counts the pages of the current document.

```
public int getPageCount()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `int` | The number of pages of a document |

Usage:

```
System.out.println("Number of pages: " + viewer.getPageCount());
```

## 5.14 zoomIn method

This method zooms to the next pre-defined level.

```
public void zoomIn() throws SignoViewerException
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.zoomIn();
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.15 zoomOut method

This method zooms to the previous pre-defined level.

```
public void zoomOut() throws SignoViewerException
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.zoomOut();
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.16 zoomFitPage method

This method zooms in until a page fills the height of the control element.

```
public void zoomFitPage() throws SignoViewerException
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.zoomFitPage();
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.17 zoomFitWidth method

This method zooms in until a page fills the width of the control element.

```
public void zoomFitWidth() throws SignoViewerException
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.zoomFitWidth();
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.18 setZoomLevel method

This method zooms to the specified level.

```
public void setZoomLevel(int zoom) throws SignoViewerException
```

| Parameter | Description |
|---|---|
| `double zoom` | zoom level as a percentage |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.setZoomLevel(66);
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.19 getZoomLevel method

This method delivers the currently set zoom level.

```
public double getZoomLevel()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `double` | The current zoom level as a percentage |

Usage:

```
System.out.println("Current zoom level: " + viewer.getZoomLevel());
```

## 5.20 scrollToPos method

This method defines a point on the current page to which the document is scrolled if this point is not currently visible.

```
public void scrollToPos(int x, int y) throws SignoViewerException
```

| Parameter | Description |
|---|---|
| `int x` | The horizontal position to be scrolled to |
| `int y` | The vertical position to be scrolled to |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.scrollToPos(100, 100);
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.21 isDocumentLoaded method

This method is able to determine whether a document is currently loaded.

```
public boolean isDocumentLoaded()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `boolean` | `true`: A document is loaded |
| | `false`: No document is loaded |

Usage:

```
if (viewer.isDocumentLoaded()) {
    System.out.println("a document is loaded");
}
```

## 5.22 getDocumentInfo method

This method requests information on the loaded document.

```
public DocumentInfoDTO getDocumentInfo()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `DocumentInfoDTO` | `DocumentInfoDTO` structure with information on the loaded document |

Usage:

```
DocumentInfoDTO doc = viewer.getDocumentInfo();
```

## 5.23 setViewerEventHandler method

This method defines a `ViewerListener`that is called as a callback when events occur in the `SignoViewer`.

```
public void setViewerEventHandler(ViewerListener handler)
```

```
public void setViewerEventHandler(ViewerListener handler, int
pageChangeThreshold)
```

| Parameter | Description |
|---|---|
| `ViewerListener handler` | Object that implements the `ViewerListener` interface |
| `int pageChangeThreshold` | Threshold (in %) that defines when the `pageChanged()` method is called; by default, the method is called when the visible height of the current page is less than the visible height of the previous or next page (50%); if the value is 0%, `pageChanged()` is called once the top edge of the next page or the entire previous page is visible; if the value is 100%, `pageChanged()` is called when the whole of the following page or the top edge of the previous page is visible. |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.setViewerEventHandler(viewerListener);
```

## 5.24 setFieldsHighlight method

This method (de)activates the highlighting of the form fields. This is deactivated by default.

```
public void setFieldsHighlight(boolean enable)
```

| Parameter | Description |
|---|---|
| `boolean enable` | `true`: activates highlighting |
| | `false`: deactivates highlighting |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.setFieldsHighlight(true);
```

## 5.25 setSignatureFieldsEnabled method

This method (de)activates empty signature fields. Deactivated signature fields are no longer highlighted and cannot be clicked any more. All signature fields are activated by default.

```
public void setSignatureFieldsEnabled(boolean enable)
```

```
public void setSignatureFieldsEnabled(String fieldName, boolean enable)
```

| Parameter | Description |
|---|---|
| `String fieldName` | (Optional) Name of signature field that is to be (de)activated. |

| | |
|---|---|
| `boolean enable` | `true`: Activates the specified or all signature fields |
| | `false`: Deactivates the specified or all signature fields |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.setSignatureFieldsEnabled("Signature1", false);
```

## 5.26 saveDocument method

This methods saves the document including the completed form fields.

```
public void saveDocument() throws CouldNotSaveException
public void saveDocument(OutputStream pdfDocument) throws
CouldNotSaveException
public void saveDocument(String fileName) throws CouldNotSaveException
```

| Parameter | Description |
|---|---|
| `OutputStream / String pdfDocument` | (Optional) `OutputStream` or relative or absolute file path in which the PDF is written; a 'Save as..' dialog box is displayed if no `OutputStream` or path is passed |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.saveDocument("example.pdf");
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.27 printDocument method

This method calls up the printer dialog box in order to print the loaded document.

```
public void printDocument() throws SignoViewerException
public void printDocument(PrinterDTO settings) throws SignoViewerException
```

| Parameter | Description |
|---|---|
| `PrinterDTO settings` | (Optional) Print settings |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    viewer.printDocument();
} catch (SignoViewerException e) {
    // error handling
}
```

## 5.28 setValidationInFields method

This method activates and deactivates syntax checks in form fields.

```
public void setValidationInFields(boolean enable)
```

| Parameter | Description |
|---|---|
| `boolean enable` | `true`: activates syntax check |
| | `false`: deactivates syntax check |
| **Return value** | **Description** |
| - | - |

Usage:

```
viewer.setValidationInFields(true);
```

## 5.29 getImage method

This method renders a PDF page in a `BufferedImage`.

### 5.29.1 Specification of a fixed size

The page is stretched to the specified size, and the original aspect ratio is not taken into account.

```
public BufferedImage getImage(int page, int width, int height)
```

| Parameter | Description |
|---|---|
| `int page` | Page (starting at 0), which is to be displayed |
| `int width` | Width in pixels |
| `int height` | Height in pixels |
| **Return value** | **Description** |
| `BufferedImage` | The page image |

Usage:

```
BufferedImage image = viewer.getImage(0, 640, 480);
```

### 5.29.2 Specification of a fixed resolution

The page is displayed at the specified resolution, the original aspect ratio is maintained.

```
public BufferedImage getImage(int page, double dpi)
```

| Parameter | Description |
|---|---|
| `int page` | Page (starting at 0), which is to be displayed |
| `double dpi` | Image resolution in dpi |
| **Return value** | **Description** |
| `BufferedImage` | The page image |

Usage:

```
BufferedImage image = viewer.getImage(0, 72);
```

## 5.30 addToolbarElement method

This method adds a user-defined element to the toolbar of the `SignoViewer` and displays the toolbar again if necessary. The element is inserted between the Zoom group and the Info group.
For the same mouseover effect as for the standard buttons, the `de.signotec.pdf.viewer.SignoViewerButtonBorderListener` class can be registered on the user-defined button as `MouseListener` (see example). The listener only works with buttons that are derived from the `javax.swing.AbstractButton` class.

```
public void addToolbarElement(Component element)
```

| Parameter | Description |
|---|---|
| `Component element` | The user-defined control element. |
| **Return value** | **Description** |
| - | - |

Usage:

```
JButton button = new JButton();
button.setBorderPainted(false);
button.setRolloverEnabled(true);
button.setFocusPainted(false);
button.setRequestFocusEnabled(false);
button.setFocusable(false);
button.addMouseListener(new SignoViewerButtonBorderListener());
button.setMargin(new Insets(5, 5, 5, 5));

viewer.addToolbarElement(button);
```

## 5.31 removeToolbarElement method

This method removes a previously added user-defined control element from the toolbar of the `SignoViewer`.

```
public void removeToolbarElement(Component element)
```

| Parameter | Description |
|---|---|
| `Component element` | The user-defined control element. |
| **Return value** | **Description** |
| - | - |

Usage:

```
JButton button = new JButton(icon);
button.setBorderPainted(false);
button.setRolloverEnabled(true);
button.setFocusPainted(false);
button.setRequestFocusEnabled(false);
button.setFocusable(false);
button.addMouseListener(new SignoViewerButtonBorderListener());
button.setMargin(new Insets(5, 5, 5, 5));

viewer.addToolbarElement(button);
```

# 6 DocumentInfoDTO class

The `de.signotec.pdf.viewer.DocumentInfoDTO` class is a data object that provides information on an opened PDF document. An object of this class is returned by the `SignoViewer.getDocumentInfo()` method.

| Method | Value | Description |
|---|---|---|
| `getTitle` | `String` | Title of the currently loaded document |
| `getAuthor` | `String` | Author of the currently loaded document |
| `getSubject` | `String` | Reference of the currently loaded document |
| `getKeywords` | `String` | Keywords of the currently loaded document |
| `getCreated` | `Date` | Creation date of the currently loaded document |
| `getModified` | `Date` | Modification date of the currently loaded document |
| `getApplication` | `String` | Name of the application with which the document was created |
| `getProducer` | `String` | Name of the application with which the document was converted |
| `getVersion` | `String` | PDF version of the currently loaded document, example: 1.4 |
| `getPages` | `int` | Number of pages of the currently loaded document |
| `isSecured` | `boolean` | Specifies whether the document is password protected |

# 7 PrinterDTO class

The `de.signotec.pdf.viewer.PrinterDTO` class is a data object containing information about how a document is printed if the `SignoViewer.printDocument()` method is called.

| Method | Value | Description |
|---|---|---|
| `getAutoRotation`<br>`setAutoRotation` | `boolean` | The document is rotated automatically where necessary; the printer settings are ignored |
| `getCenterInPage`<br>`setCenterInPage` | `boolean` | The document is centred on the page if it is smaller than the page |
| `getExpandToMargins`<br>`setExpandToMargins` | `boolean` | The document is enlarged to the size of the page if it is smaller than the page |
| `getShrinkToMargins`<br>`setShrinkToMargins` | `boolean` | The document is shrunk to the size of the page if it is larger than the page |
| `getSystemPrintDialog`<br>`setSystemPrintDialog` | `boolean` | Establishes which print dialog to use.<br>`true`: Native operating system dialog<br>`false`:Built-in application dialog<br>The built-in dialog is used by default. |

# 8 Enum ToolBarEnum

The `de.signotec.pdf.viewer.enums.ToolbarEnum` enum is used to display bitmasks in a more user-friendly manner. This enumeration is used as a parameter by the `signoViewer.configureToolbar()` method.

| Value | Description |
|---|---|
| `OPENDOC` | Open button |
| `SAVEDOC` | Save document |
| `CLOSEDOC` | Close document |
| `PRINT` | Print button |
| `PREVPAGE` | Previous page |
| `PAGENUMBER` | Current page |
| `FIRSTPAGE` | First page |
| `NEXTPAGE` | Next page |
| `LASTPAGE` | Last page |
| `FITTOWIDTH` | Fit to page |
| `FITTOPAGE` | Fit to width |
| `ZOOMOUT` | Reduce |
| `SCALEFAC` | Current zoom factor |
| `ZOOMIN` | Enlarge |
| `INFO` | Info button |
| `LOGO` | Logo left-aligned |
| `LOGO_RIGHT` | Logo right-aligned |
| `GROUPDOC` | Combination of `OPENDOC`, `SAVEDOC`, `CLOSEDOC` and `PRINT` |
| `GROUPPAGE` | Combination of `FIRSTPAGE`, `PREVPAGE`, `PAGENUMBER`, `NEXTPAGE` and `LASTPAGE` |
| `GROUPSCALE` | Combination of `FITTOWIDTH` and `FITTOPAGE` |
| `GROUPZOOM` | Combination of `ZOOMOUT`, `SCALEFAC` and `ZOOMIN` |
| `GROUPINFO` | Combination of `INFO` and `LOGO` |
| `ALL` | Combination of all values except `LOGO_RIGHT` |

# 9 ViewerListener interface

The `de.signotec.pdf.viewer.events.ViewerListener` interface is intended as a callback interface that is called if events occur in the `SignoViewer`.

## 9.1 pageChanged method

This method is called when the actual page number has changed.

```
public void pageChanged(int pageNumber)
```

| Parameter | Description |
|---|---|
| `int pageNumber` | Number of the currently displayed page |
| **Return value** | **Description** |
| - | - |

Usage:

```
public void pageChanged(int pageNumber) {
    System.out.println("Current page: " + pageNumber);
}
```

## 9.2 openClicked method

This method is called when the 'Open' button on the toolbar is clicked.

```
public boolean openClicked()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default dialog box should be opened |

Usage:

```
public boolean openClicked() {
    return false;
}
```

## 9.3 documentOpened method

This method is called when a document has been opened by clicking the 'Open' button on the toolbar.

```
public void documentOpened(File file)
```

| Parameter | Description |
|---|---|
| `File file` | File that has been opened |
| **Return value** | **Description** |
| - | - |

Usage:

```
public void documentOpened(File file) {
    System.out.println(file.getAbsolutePath() + " opened");
}
```

## 9.4 closeClicked method

This method is called when the 'Close' button on the toolbar is clicked.

```
public boolean closeClicked()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default dialog box should be opened |

Usage:

```
public boolean closeClicked() {
    return false;
}
```

## 9.5 documentClosed method

This method is called when a document has been closed by clicking the 'Close' button on the toolbar.

```
public void documentClosed()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| - | - |

Usage:

```
public void documentClosed() {
    System.out.println("document opened");
}
```

## 9.6 saveClicked method

This method is called when the 'Save' button on the toolbar is clicked.

```
public boolean saveClicked()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default dialog box should be opened |

Usage:

```
public boolean saveClicked() {
   return false;
}
```

## 9.7 documentSaved method

This method is called when a document has been saved by clicking the 'Save' button on the toolbar.

```
public void documentOpened(File file)
```

| Parameter | Description |
|---|---|
| File file | File that has been saved |
| **Return value** | **Description** |
| - | - |

Usage:

```
public void documentSaved(File file) {
    System.out.println(file.getAbsolutePath() + " saved");
}
```

## 9.8 printClicked method

This method is called when the 'Print' button on the toolbar is clicked.

```
public boolean printClicked()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| boolean | true: a built-in functionality has been implemented |
| | false: Default dialog box should be opened |

Usage:

```
public boolean printClicked() {
    PrinterDTO options = new PrinterDTO();
    options.setShrinkToMargins(false);

    try {
        this.viewer.printDocument(options);
    } catch (SignoViewerException e) {
        // handle error
    }

    return true;
}
```

## 9.9 signatureFieldMouseClicked method

This method is called when a signature field is clicked with a mouse button.

```
public boolean signatureFieldMouseClicked(String fieldName, int pageNumber,
int buttonNumber)
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `int pageNumber` | Page number of the signature field |
| `int buttonNumber` | `1` : Left mouse button |
| | `2`: Middle mouse button |
| | `3`: Right mouse button |
| **Return value** | **Description** |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default dialog box should be opened |

Usage:

```
public boolean signatureFieldMouseClicked(String fieldName,
        int pageNumber, int buttonNumber) {

    System.out.println("On page : " + (pageNumber + 1)
            + " field " + fieldName + " clicked ");

    return true;
}
```

## 9.10 signatureFieldMouseEntered method

This method is called if a `MouseEntered` event has been triggered on a signature field.

```
public boolean signatureFieldMouseEntered(String fieldName, int pageNumber,
int buttonNumber)
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `int pageNumber` | Page number of the signature field |
| `int buttonNumber` | `1` : Left mouse button |
| | `2`: Middle mouse button |
| | `3`: Right mouse button |
| **Return value** | **Description** |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default |

Usage:

```
public boolean signatureFieldMouseEntered(String fieldName,
        int pageNumber) {

    return false;
}
```

## 9.11 signatureFieldMouseExited method

This method is called if a `MouseExited` event has been triggered on a signature field.

```
public boolean signatureFieldMouseExited(String fieldName, int pageNumber, int
buttonNumber)
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `int pageNumber` | Page number of the signature field |
| `int buttonNumber` | `1` : Left mouse button |
| | `2`: Middle mouse button |
| | `3`: Right mouse button |
| **Return value** | **Description** |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default |

Usage:

```
public boolean signatureFieldMouseExited(String fieldName,
        int pageNumber) {

    return false;
}
```

## 9.12 signatureFieldMouseReleased method

This method is called if a `MouseReleased` event has been triggered on a signature field.

```
public boolean signatureFieldMouseReleased(String fieldName, int pageNumber,
int buttonNumber);
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `int pageNumber` | Page number of the signature field |
| `int buttonNumber` | `1` : Left mouse button |
| | `2`: Middle mouse button |
| | `3`: Right mouse button |
| **Return value** | **Description** |
| `boolean` | `true`: the shortcut menu is not displayed |
| | `false`: Default |

Usage:

```
public boolean signatureFieldMouseReleased(String fieldName,
        int pageNumber) {

    return false;
}
```

## 9.13 signatureFieldMousePressed method

This method is called if a `MousePressed` event has been triggered on a signature field.

```
public boolean signatureFieldMousePressed(String fieldName, int pageNumber,
int buttonNumber)
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `int pageNumber` | Page number of the signature field |

| | |
|---|---|
| `int buttonNumber` | `1` : Left mouse button |
| | `2`: Middle mouse button |
| | `3`: Right mouse button |
| Return value | Description |
| `boolean` | `true`: a built-in functionality has been implemented |
| | `false`: Default |

Usage:

```
public boolean signatureFieldMousePressed( String fieldName,
        int pageNumber) {

    return false;
}
```

# 10 SignoPdfSigner class

Another main component is the `de.signotec.pdf.signer.SignoPdfSigner` class with its derived `de.signotec.pdf.signer.SignoPdfSignerSTPad` class. These classes provide methods for reading information on signature fields in PDF documents and for signing these fields.
The `SignoPdfSigner` class should only be instantiated if interdigitation with the available signoPAD API is not required, for example, because signature capture is to take place on a different system to that on which the document is signed (client/server solution). If capture and signing are to take place on the same system, the `SignoPdfSignerSTPad` class should always be used for security reasons.

This `SignoPdfSigner` class includes two public constructors:

```
SignoPdfSigner signer = null;
try {
    // 1. pass path of the document
    signer = new SignoPdfSigner("test.pdf");
    // 2. pass stream of the document
    signer = new SignoPdfSigner(inStream);
} catch (SignoSignerException e) {
    // error handling
}
```

From now on, the typical process of PDF signature using the `SignoPdfSigner` will be referred to as follows:

```
// initialize SignoPdfSigner
SignoPdfSigner signer = new SignoPdfSigner("test.pdf");
// sign the document with data collected elsewhere
signer.signDocument("sig1", sigData, "sign.ks", "secret", "secret2",
        "sign", "bio.ks", "secret3", "bio", "signed.pdf", bioData,
        signImage, padId, padType, padFirmware);
// add more signatures with signDocument()
```

## 10.1 signDocument method

This method is used to digitally sign the loaded document. It is not necessary to output the document if further signatures are to be captured.
This method can be called up multiple times for a `SignoPdfSigner` object.

### 10.1.1 Encryption in the signature device

When a pad in which a public key for encryption is stored is connected, the biometric data can be encrypted in the device. Consequently, a public RSA key is not needed on the PC and the biometric data is already encrypted before transfer to the PC.
This technology is currently only supported in Windows.

```
public void signDocument(String fieldName, SigningDTO signData,
     PrivateKey signingKey, Certificate[] signingCertChain, String bioCertRef,
     OutputStream output, String bioData, BufferedImage signImage,
     String padId, String padType, String padVersion)
     throws NoSignatureFieldException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |

| | |
|---|---|
| `PrivateKey signingKey` | Private key for signing |
| `Certificate[] signingCertChain` | Certificate chain for signing |
| `String bioCertRef` | ID of the RSA key used, for instance the string returned by `SigPadApi.getEncryptionCertId()` |
| `OutputStream output` | Stream in which the PDF document is written; may be `null` |
| `String bioData` | Base 64-coded biometric data of the signature in SignData format encrypted in the signature device |
| `BufferedImage signImage` | Image of the captured signature |
| `String padId` | ID of the signature device used to capture the signature, usually the serial number |
| `String padType` | Type or name of the signature device used to capture the signature |
| `String padVersion` | Firmware version of the signature device used to capture the signature |
| Return value | Description |
| – | - |

Usage:

```
File storeFile = new File("myKeystore.ks");
char[] storePwd = "myStorePassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_JAVA_KEYSTORE;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, storePwd);

char[] keyPwd = "myKeyPassword".toCharArray();
PrivateKey signingKey = loader.getPrivateKey("signAlias", keyPwd);
Certificate[] signingCertChain = loader.getCertificateChain("signAlias");

signer.signDocument("sig1", sigData, signingKey, signingCertChain,
        bioCertRef, null, bioData, signImage, padId, padType,
        padFirmware);
```

### 10.1.2 Encryption and signing of the biometric data in the signature device

When a pad containing a public key for encryption and a key pair for signing is connected, the biometric data can be encrypted and signed in the device. Consequently, a public RSA key is not needed on the PC and the biometric data is already encrypted before transfer to the PC. It is also possible to subsequently verify the integrity of the biometric data.
This technology is currently only supported in Windows.

```
public void signDocument(String fieldName, SigningDTO signData,
     PrivateKey signingKey, Certificate[] signingCertChain, String bioCertRef,
     OutputStream output, HashType padSignatureType,
     RSAScheme padSignatureScheme, byte[] padSignature,
     X509Certificate padSigningCert, String bioData, BufferedImage signImage,
     String padId, String padType, String padVersion)
     throws NoSignatureFieldException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |
| `PrivateKey signingKey` | Private key for signing |

| | |
|---|---|
| `Certificate[] signingCertChain` | Certificate chain for signing |
| `String bioCertRef` | ID of the RSA key used, for instance the string returned by `SigPadApi.getEncryptionCertId()` |
| `OutputStream output` | Stream in which the PDF document is written; may be `null` |
| `HashType padSignatureType` | Hash type used to calculate the `padSignature` (also see the signoPAD-API documentation) |
| `RSAScheme padSignatureScheme` | RSA schema used to calculate the `padSignature` (also see the signoPAD-API documentation) |
| `byte[] padSignature` | Digital signature using hash 1 and/or hash 2 (also see `SigPadApi.signHash()` in the signoPAD-API documentation) |
| `X509Certificate padSigningCert` | Public certificate that allows the `padSignature` to be verified (also see `SigPadApi.getSigningCert()` in the signoPAD-API documentation) |
| `String bioData` | Base 64-coded biometric data of the signature in SignData format encrypted in the signature device |
| `BufferedImage signImage` | Image of the captured signature |
| `String padId` | ID of the signature device used to capture the signature, usually the serial number |
| `String padType` | Type or name of the signature device used to capture the signature |
| `String padVersion` | Firmware version of the signature device used to capture the signature |
| Return value | Description |
| - | - |

Usage:

```
File storeFile = new File("myKeystore.ks");
char[] storePwd = "myStorePassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_JAVA_KEYSTORE;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, storePwd);

char[] keyPwd = "myKeyPassword".toCharArray();
PrivateKey signingKey = loader.getPrivateKey("signAlias", keyPwd);
Certificate[] signingCertChain = loader.getCertificateChain("signAlias");

signer.signDocument("sig1", sigData, signingKey, signingCertChain,
        bioCertRef, null, HashType.COMBINATION, RSAScheme.PSS, signature,
        cert, bioData, signImage, padId, padType, padFirmware);
```

### 10.1.3 Encryption on the PC

This functionality is supported by all signature devices.

```
public void signDocument(String fieldName, SigningDTO signData,
     PrivateKey signingKey, Certificate[] signingCertChain,
     X509Certificate bioCert, OutputStream output, String bioData,
     BufferedImage signImage, String padId, String padType, String padVersion)
     throws NoSignatureFieldException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |

| | |
|---|---|
| `PrivateKey signingKey` | Private key for signing |
| `Certificate[] signingCertChain` | Certificate chain for signing |
| `X509Certificate bioCert` | Certificate with the public key for encrypting the biometric data |
| `OutputStream output` | Stream in which the PDF document is written; may be `null` |
| `String bioData` | Base64-coded biometric data of the signature in SignData format |
| `BufferedImage signImage` | Image of the captured signature |
| `String padId` | ID of the signature device used to capture the signature, usually the serial number |
| `String padType` | Type or name of the signature device used to capture the signature |
| `String padVersion` | Firmware version of the signature device used to capture the signature |
| Return value | Description |
| – | - |

Usage:

```
File storeFile = new File("myKeystore.ks");
char[] storePwd = "myStorePassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_JAVA_KEYSTORE;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, storePwd);

char[] keyPwd = "myKeyPassword".toCharArray();
PrivateKey signingKey = loader.getPrivateKey("signAlias", keyPwd);
Certificate[] signingCertChain = loader.getCertificateChain("signAlias");
X509Certificate bioCert = loader.getX509Certificate("bioAlias");

signer.signDocument("sig1", sigData, signingKey, signingCertChain,
        bioCert, null, bioData, signImage, padId, padType, padFirmware);
```

## 10.2 getReferenceCount method

This method returns the number of signature fields contained in a PDF document. It is a static method that can be used independently of the signing.

```
public static int getReferenceCount(InputStream pdfDocument,
     boolean filledOnly) throws SignoSignerException

public static int getReferenceCount(InputStream pdfDocument, int page,
     boolean filledOnly) throws SignoSignerException

public static int getReferenceCount(String pdfDocument, boolean filledOnly)
     throws SignoSignerException

public static int getReferenceCount(String pdfDocument, int page,
     boolean filledOnly) throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `InputStream / String pdfDocument` | `InputStream` or path and file name of a PDF document |
| `int page` | (Optional) Page on which signature fields are searched |
| `boolean filledOnly` | `true`: Only completed signature fields are counted |
| | `false`: All existing signature fields are counted |

| Return value | Description |
|---|---|
| `int` | Number of found signature fields |

Usage:

```
try {
    int sig = SignoPdfSigner.getReferenceCount("test.pdf", false);
    System.out.println("There are " + sig + " signature fields");

    sig = SignoPdfSigner.getReferenceCount("test.pdf", 2, true);
    System.out.println("There are " + sig + " signatures on page 2");
} catch (SignoSignerException e) {
    // error handling
}
```

## 10.3 getBiproSignaturMetaData method

This method returns a `BiproSignaturMetaDataDTO` object that contains the BiPRO-compliant metadata of a signature.
It is a static method that can be used independently of the signing.

```
public static BiproSignaturMetaDataDTO getBiproSignaturMetaData(
     InputStream pdfDocument, String fieldName) throws SignoSignerException

public static BiproSignaturMetaDataDTO getBiproSignaturMetaData(
     String pdfDocument, String fieldName) throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `InputStream / String pdfDocument` | `InputStream` or path and file name of a PDF document |
| `String fieldName` | Name of the signature field |
| **Return value** | **Description** |
| `BiproSignatureMeta-DataDTO` | Signature meta data according to BiPRO-standard |

Usage:

```
try {
    BiproSignaturMetaDataDTO meta =
            SignoPdfSigner.getBiproSignaturMetaData("test.pdf", "sig1");
    System.out.println(meta.getBiproSignaturSoftwareHersteller());
} catch (SignoSignerException e) {
    // error handling
}
```

## 10.4 getBiometricData method

This method returns a `BiometricDataDTO` object containing the unencrypted biometric data that was saved in a signature field. To call this method, the certificate with the private key is necessary to decrypt the biometric data.
It is a static method that can be used independently of the signing.

```
public static BiometricDataDTO getBiometricData(String pdfDocument,
     String fieldName, PrivateKey bioKey)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `InputStream pdfDocument` | `InputStream` of a PDF document |
| `String fieldName` | Name of the signature field |
| `PrivateKey bioKey` | Private RSA key for decrypting the biometric data |
| **Return value** | **Description** |
| `BiometricDataDTO` | Biometric data |

Usage:

```
File storeFile = new File("myKey.p12");
char[] password = "myPassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_PKCS12;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, password);
PrivateKey bioKey = loader.getFirstPrivateKey(password);

BiometricDataDTO bio = SignoPdfSigner.getBiometricData(
        document, "sig1", bioKey);

System.out.println("Biometric: " + bio.getBioData());
System.out.println("Hash: " + bio.getDocHashValue());
System.out.println("RecalcedHash: " + bio.getDocRecalcedHashValue());

if (!bio.getDocHashValue().equals(bio.getDocRecalcedHashValue())) {
    System.out.println("-> DOCUMENT WAS CHANGED!");
}
```

## 10.5 verifyPdfDocument method

This method tests the signatures of a document for validity.
It is a static method that can be used independently of the signing.

```
public static boolean verifyPdfDocument(InputStream pdfDocument)
     throws SignoSignerException

public static boolean verifyPdfDocument(String pdfDocument)
     throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `InputStream / String pdfDocument` | `InputStream` or path and file name of a PDF document |
| **Return value** | **Description** |
| `boolean` | `true`: All signatures are valid |
| | `false`: Invalid signatures were found |

Usage:

```
try {
    if (SignoPdfSigner.verifyPdfDocument("test.pdf")) {
        System.out.println("valid");
    } else {
        System.out.println("INVALID");
    }
} catch (SignoSignerException e) {
    // error handling
}
```

## 10.6 getSignatureFieldInfo method

This method returns an array of `SignatureFieldInfoDTO` objects that contain specifications on a signature field.
It is a static method that can be used independently of the signing.

```
public static SignatureFieldInfoDTO[] getSignatureFieldInfo(
     InputStream pdfDocument) throws SignoSignerException

public static SignatureFieldInfoDTO[] getSignatureFieldInfo(
     InputStream pdfDocument, boolean emptyOnly) throws SignoSignerException

public static SignatureFieldInfoDTO[] getSignatureFieldInfo(
     InputStream pdfDocument, boolean emptyOnly, boolean useFieldOrder)
     throws SignoSignerException

public static SignatureFieldInfoDTO[] getSignatureFieldInfo(
     String pdfDocument) throws SignoSignerException

public static SignatureFieldInfoDTO[] getSignatureFieldInfo(
     String pdfDocument, boolean emptyOnly) throws SignoSignerException

public static SignatureFieldInfoDTO[] getSignatureFieldInfo(
     String pdfDocument, boolean emptyOnly, boolean useFieldOrder)
     throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `InputStream / String pdfDocument` | `InputStream` or path and file name of a PDF document |
| `boolean emptyOnly` | `true`: Information is returned on unsigned signature fields only (optional) |
| | `false`: Information is returned on all signature fields (default) |
| `boolean useFieldOrder` | `true`: The returned array is sorted in the order in which the fields were added (optional) |
| | `false`: The returned array is sorted in the tab order of the fields (default) |
| **Return value** | **Description** |
| `SignatureFieldInfo-DTO[]` | Array of data objects with field information. |

Usage:

```
try {
    SignatureFieldInfoDTO[] infos =
            SignoPdfSigner.getSignatureFieldInfo("test.pdf");

    for (SignatureFieldInfoDTO field : infos) {
        // process data
    }
} catch (SignoSignerException e) {
    // error handling
}
```

## 11 SignoPdfSignerSTPad class

The `de.signotec.pdf.signer.SignoPdfSignerSTPad` class is used to capture the signature using a signotec Sigma, Omega, Gamma, Delta or Alpha signing pad and requires an object of the type `de.signotec.STPad.api.SigPadApi` (see the section Additional components). It is recommended to use this class to capture a signature, as it allows all advanced functions of signotec LCD signature pads to be used.

This class includes two public constructors:

```
SignoPdfSignerSTPad signer = null;

try {
    // 1. pass the path of the document
    signer = new SignoPdfSignerSTPad("test.pdf", padApi);
    // 2. Pass the stream of the document
    signer = new SignoPdfSignerSTPad(inStream, padApi);
} catch (SignoSignerException e) {
    // error handling
}
```

From now on, the typical process of the PDF signature with the SignoPdfSignerSTPad is referred to as follows:

```
SigPadFacade facade = SigPadFacade.getInstance();
facade.initializeApi();
SigPadDevice[] devices = facade.getSignatureDevices();

if (devices.length > 0) {
    SigningDTO sigData = new SigningDTO();
    /* fill the SigningDTO instance */
    // search for devices and open a connection
    SigPadApi padApi = new SigPadApi(devices[0]);
    padApi.openDevice(null, null);

    /* Create Hotspots and signing area on the signature device
       (see also signoPAD-API Java Documentation) */

    // initialize SignoPdfSignerSTPad
    SignoPdfSignerSTPad signer =
            new SignoPdfSignerSTPad("test.pdf", pad);
    // start signature capture
    signer.initSignature("sig1", sigData, signingKey, signingCertChain,
            bioCert);

    /* wait until signature is captured */

    // finish capturing and save the document
    signer.confirmSignature("signed.pdf");

    /* capture more with initSignature and confirmSignature */

    // close the connection
    pad.closeDevice();
}
facade.finalizeApi();
```

## 11.1 displaySignatureField method

This method displays the area around a signature field in a PDF as an image on the signature device's display so that a signature can be provided directly in the document.

```
public SigPadRectangle displaySignatureField(String fieldName,
     Color highlightColor, int margin, SignoViewer viewer,
     boolean drawInBackground) throws NoSignatureFieldException,
     InvalidParameterException, SignoSignerException

public SigPadRectangle displaySignatureField(SigningDTO signatureField,
     Color highlightColor, int margin, SignoViewer viewer,
     boolean drawInBackground)
     throws InvalidParameterException, SignoSignerException

public SigPadRectangle displaySignatureField(String fieldName,
     Color highlightColor, RectangleDTO margins, SignoViewer viewer,
     boolean drawInBackground) throws NoSignatureFieldException,
     InvalidParameterException, SignoSignerException

public SigPadRectangle displaySignatureField(SigningDTO signatureField,
     Color highlightColor, RectangleDTO margins, SignoViewer viewer,
     boolean drawInBackground)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of signature field that is to be displayed |
| `SigningDTO signatureField` | Data object containing the page and signature field rectangle that is to be displayed |
| `Color highlightColor` | Colour in which the signature field is to be highlighted on the display; may be `null` |
| `int margin` | Minimum distance of the signature field from the display edges in pixels; if the value is less than 0, the document will be shown in original size |
| `RectangleDTO margins` | Minimum distances of the signature field from the display edges in pixels; if the value is `null`, the document will be shown in original size |
| `SignoViewer viewer` | `SignoViewer` instance that has loaded the document to be displayed; if the value is `null`, a new instance will be generated and the document that has been passed to the constructor of the current `SignoPdfSignerSTPad` instance will be used |
| `boolean drawInBackground` | If the value is `true`, the document will only be loaded in the background buffer of the pad and not displayed; details on image memory can be found in the signoPAD-API Java documentation |
| **Return value** | **Description** |
| `SigPadRectangle` | The signature field in the coordinates as it is shown on the display; these coordinates can be used to define the signature window |

Usage:

```
SigPadRectangle rect;
try {
    rect = signer.displaySignatureField("sig1", new Color(0,0,255,100),
            20, this.viewer, false);
} catch (SignoSignerException e) {
    // error handling
}
```

## 11.2 displaySignatureFields method

This method displays a page of a PDF document containing one or more signature fields rendered as an image on the display of the signature device so that a signature can be provided directly in the document. The page is always shown in original size, meaning that content may be cut off depending on page and display sizes.

```
public SigPadRectangle[] displaySignatureFields(String[] fieldNames,
     Color highlightColor, ImageAlign align, SignoViewer viewer,
     boolean drawInBackground) throws NoSignatureFieldException,
     InvalidParameterException, SignoSignerException
```

```
public SigPadRectangle[] displaySignatureFields(SigningDTO[] signatureFields,
     Color highlightColor, ImageAlign align, SignoViewer viewer,
     boolean drawInBackground)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String[] fieldNames` | Name of one or more signature fields, all of which must be on the same page. |
| `SigningDTO[] signatureFields` | Data objects containing the page and the rectangles of one or more signature fields |
| `Color highlightColor` | Colour in which the signature field is to be highlighted on the display; may be `null` |
| `ImageAlign align` | Alignment of the page on the display |
| `SignoViewer viewer` | `SignoViewer` instance that has loaded the document to be displayed; if the value is `null`, a new instance will be generated and the document that has been passed to the constructor of the current `SignoPdfSignerSTPad` instance will be used |
| `boolean drawInBackground` | If the value is `true`, the document will only be loaded in the background buffer of the pad and not displayed; details on image memory can be found in the signoPAD-API Java documentation |
| **Return value** | **Description** |
| `SigPadRectangle[]` | The signature fields in the coordinates as they are shown on the display; these coordinates can be used to define the signature windows |

Usage:

```
SigPadRectangle[] rects;
try {
    String[] fieldNames = new String[2];
    fieldNames[0] = "sig1";
    fieldNames[1] = "sig2";
    rects = signer.displaySignatureFields(fieldNames,
            new Color(0,0,255,100), ImageAlign.BOTTOMCENTER,
            this.viewer, false);
} catch (SignoSignerException e) {
    // error handling
}
```

## 11.3 initSignature method

This method is used to start the capture of one or more signatures. The `startSignature()` method of the `SigPadApi` object transferred in the constructor is triggered at this point. If one of the specified signature fields is not available, it is created. In this case, the specifications `Rectangle` and `Page` of the `SigningDTO` object must be set.
When using an Alpha pad, several fields on a page can be signed at once by defining a signature window for each field using `SigPadApi.setSignRects()`. The indices of the signature window must then correspond to the indices of the `fieldNames` and `signData` parameters. The `confirmSignature()` method must not be called until all signatures have been captured.

### 11.3.1 Signing and encrypting in the signature device

When a pad in which a private key for signing and a public key for encryption are stored is connected, the device can be used for digitally signing the document and for encrypting the biometric data. Consequently, no RSA keys are needed on the PC and the biometric data is already encrypted before transfer to the PC.
This technology is currently only supported in Windows.

```
public void initSignature(String fieldName, SigningDTO signData)
     throws InvalidParameterException, SignoSignerException

public void initSignature(String[] fieldNames, SigningDTO[] signData)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `String[] fieldNames` | Array with names of several signature fields |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |
| `SigningDTO[] signData` | Array with data objects with all specifications necessary for the signature |
| **Return value** | **Description** |
| - | - |

Usage:

```
SigningDTO data = new SigningDTO();
try {
    // fill the SigningDTO instance
    signer.initSignature("sig1", data);
} catch (SignoSignerException e) {
    // error handling
}
```

### 11.3.2 Signing in the signature device, encrypting on a PC

When a pad in which a private key for signing is stored is connected, the document can be digitally signed on the device. Consequently, a private RSA key is not required on the PC. This technology is currently only supported in Windows.

```
public void initSignature(String fieldName, SigningDTO signData,
     X509Certificate bioCert)
     throws InvalidParameterException, SignoSignerException

public void initSignature(String[] fieldNames, SigningDTO[] signData,
     X509Certificate bioCert)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `String[] fieldNames` | Array with names of several signature fields |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |
| `SigningDTO[] signData` | Array with data objects with all specifications necessary for the signature |
| `X509Certificate bioCert` | Certificate for the encryption of biometric data |
| **Return value** | **Description** |
| – | - |

Usage:

```
File storeFile = new File("myKeystore.ks");
char[] storePwd = "myStorePassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_JAVA_KEYSTORE;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, storePwd);

X509Certificate bioCert = loader.getX509Certificate("bioAlias");
SigningDTO data = new SigningDTO();

signer.initSignature("sig1", data, bioCert);
```

### 11.3.3 Signing on a PC, encrypting on the signature device

When a pad in which a public key for encryption is stored is connected, the biometric data can be encrypted in the device. Consequently, a public RSA key is not needed on the PC and the biometric data is already encrypted before transfer to the PC.
This technology is currently only supported in Windows.

```
public void initSignature(String fieldName, SigningDTO signData,
     PrivateKey signingKey, Certificate[] signingCertChain)
     throws InvalidParameterException, SignoSignerException

public void initSignature(String[] fieldNames, SigningDTO[] signData,
     PrivateKey signingKey, Certificate[] signingCertChain)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `String[] fieldNames` | Array with names of several signature fields |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |
| `SigningDTO[] signData` | Array with data objects with all specifications necessary for the signature |

| | |
|---|---|
| `PrivateKey signingKey` | Private key for signing |
| `Certificate[] signingCertChain` | Certificate chain for signing |
| Return value | Description |
| - | - |

Usage:

```
File storeFile = new File("myKeystore.ks");
char[] storePwd = "myStorePassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_JAVA_KEYSTORE;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, storePwd);

char[] keyPwd = "myKeyPassword".toCharArray();
PrivateKey signKey = loader.getPrivateKey("signAlias", keyPwd);
Certificate[] signCertChain = loader.getCertificateChain("signAlias");

SigningDTO data = new SigningDTO();
signer.initSignature("sig1", data, signKey, signCertChain);
```

### 11.3.4 Signing and encrypting on a PC

This functionality is supported by all signature devices. In order to use it, a private key for signing and a public key for encryption must be available on the PC.

```
public void initSignature(String fieldName, SigningDTO signData,
     PrivateKey signingKey, Certificate[] signingCertChain,
     X509Certificate bioCert)
     throws InvalidParameterException, SignoSignerException
```

```
public void initSignature(String[] fieldNames, SigningDTO[] signData,
     PrivateKey signingKey, Certificate[] signingCertChain,
     X509Certificate bioCert)
     throws InvalidParameterException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `String[] fieldNames` | Array with names of several signature fields |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |
| `SigningDTO[] signData` | Array with data objects with all specifications necessary for the signature |
| `PrivateKey signingKey` | Private key for signing |
| `Certificate[] signingCertChain` | Certificate chain for signing |
| `X509Certificate bioCert` | Certificate with the public key for encrypting the biometric data |
| Return value | Description |
| - | - |

Usage:

```
File storeFile = new File("myKeystore.ks");
char[] storePwd = "myStorePassword".toCharArray();
String storeType = KeyLoader.STORE_TYPE_JAVA_KEYSTORE;
KeyLoader loader = KeyLoader.getInstance(storeFile, storeType, storePwd);

char[] keyPwd = "myKeyPassword".toCharArray();
PrivateKey signKey = loader.getPrivateKey("signAlias", keyPwd);
Certificate[] signCertChain = loader.getCertificateChain("signAlias");
X509Certificate bioCert = loader.getX509Certificate("bioAlias");

SigningDTO data = new SigningDTO();
signer.initSignature("sig1", data, signKey, signCertChain, bioCert);
```

## 11.4 confirmSignature method

This method ends the capture of one or more signatures. The `confirmSignature()` and `getSignatureData()` methods of the `SigPadApi` object transferred in the constructor are triggered at this point. The signature data is subsequently no longer available in the `SigPadApi` object.
This method can only be called up once after executing the `initSignature()` method. The `output` parameter is not required if the document is not to be saved yet, for example, because other signatures are still to be captured.
The `initSignature()` and `confirmSignature()` methods can be called up multiple times for a `SignoPdfSignerSTPad` object.
The `type` and `scheme` parameters can only be passed if the signature device is used to encrypt the biometric data (see also `initSignature()`) and a private key for signing is also saved there.

```
public int[] confirmSignature() throws NoCapturingException,
     NoSignatureFieldException, SignoSignerException

public int[] confirmSignature(OutputStream output)
     throws NoCapturingException, NoSignatureFieldException,
     SignoSignerException

public int[] confirmSignature(String output) throws NoCapturingException,
     NoSignatureFieldException, SignoSignerException

public int[] confirmSignature(OutputStream output, int minNumPoints)
     throws NoCapturingException, SignatureTooShortException,
     NoSignatureFieldException, SignoSignerException

public int[] confirmSignature(String output, int minNumPoints)
     throws SignoSignerException NoCapturingException,
     SignatureTooShortException, NoSignatureFieldException,
     SignoSignerException

public int[] confirmSignature(OutputStream output, int minNumPoints,
     HashType type, RSAScheme scheme) throws NoCapturingException,
     SignatureTooShortException, NoSignatureFieldException,
     SignoSignerException

public int[] confirmSignature(String output, int minNumPoints, HashType type,
     RSAScheme scheme)
     throws NoCapturingException, SignatureTooShortException,
     NoSignatureFieldException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `OutputStream /String output` | (Optional) `OutputStream` or relative or absolute file path in which the PDF document is written |

| | |
|---|---|
| `int minNumPoints` | (Optional) Minimum number of points that each captured signature must contain (default is 0) |
| `HashType type` | (Optional) Defines which data is to be signed in the signature device and embedded in the biometric data; as a rule, `COMBINATION` should be used where the signed document is inseparably linked to the biometric data; if `type` is not passed, no digital signature is embedded |
| `RSAScheme scheme` | (Optional) RSA scheme with which the digital signature, which is embedded in the biometric data, is to be made; if `scheme` is not passed, no digital signature is embedded |
| **Return value** | **Description** |
| `int[]` | Array with the number of captured points of the individual signatures; the size and indices of the array correspond to the `fieldNames` parameter of the `initSignature()` method |

Usage:

```
try {
    signer.confirmSignature("signed.pdf", 50);
} catch (SignoSignerException e) {
    // error handling
}
```

## 11.5 signDocument method

This method is used to digitally sign the loaded document in the signature device without capturing a signature (therefore also without biometric data). Usually this is used for a final document signature and an empty rectangle (all coordinates 0) is passed to the SigningDTO object so that no visual signature field is displayed.
This method can be called up multiple times for a `SignoPdfSignerSTPad` object.
This technology is currently only supported in Windows.

```
public void signDocument(String fieldName, SigningDTO signData,
     OutputStream output)
     throws NoSignatureFieldException, SignoSignerException

public void signDocument(String fieldName, SigningDTO signData, String output)
     throws NoSignatureFieldException, SignoSignerException
```

| Parameter | Description |
|---|---|
| `String fieldName` | Name of the signature field |
| `SigningDTO signData` | Data object with all specifications necessary for the signature |
| `OutputStream /String output` | (Optional) `OutputStream` or relative or absolute file path in which the PDF document is written; may be `null` |
| **Return value** | **Description** |
| - | - |

Usage:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
try {
    SigningDTO signingDto = new SigningDTO();
    signingDto.setPage(1);
    signingDto.setRectangle(new RectangleDTO(0, 0, 0, 0));
    signingDto.setTimestampFormat(null);
    signer.signDocument("FinalSignature", signingDto, baos);
} catch (SignoSignerException e) {
```

```
    // error handling
}
```

## 11.6 getDocumentDisplay method

This method creates a predefined configuration for displaying a document on the signature device. The configuration refers to the document in a `SignoViewer` and can be sent to the signature device with `displayDocument()`.
The settings contain every page of the document.

```
public DocumentDisplayDTO getDocumentDisplay(SignoViewer viewer)
     throws SignoSignerException

public DocumentDisplayDTO getDocumentDisplay(SignoViewer viewer,
     float resolution) throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `SignoViewer viewer` | Control element with loaded document. |
| `float resolution` | (Optional) The resolution, in PPI, used in displaying the document on the signature device. The document is displayed in its original size (72 PPI) if no resolution is specified. |
| **Return value** | **Description** |
| `DocumentDisplayDTO` | Display settings for the document. |

Usage:

```
try {
    SignoPdfSignerSTPad signer =
            new SignoPdfSignerSTPad("input.pdf" , padApi);
    // create the display configuration with all pages
    DocumentDisplayDTO doc = signer.getDocumentDisplay(viewer);
    // display the document on the signature pad
    signer.displayDocument(viewer, doc);
} catch (SignoSignerException e) {
    // error handling
}
```

## 11.7 displayDocument method

This method applies a display configuration (`getDocumentDisplay()`) to the signature device. It fills the display with the contents of the first page and then returns. The rest of the document is transferred to the device in the background in order to minimise the waiting time for the user. If the user scrolls on the pad while the display is being loaded, the newly visible image section is automatically loaded first.

This method should only be used with a signotec Delta device and throws an error if the device does not have sufficient image memory.

```
public LoadImageFuture displayDocument(SignoViewer viewer,
     DocumentDisplayDTO documentDisplay) throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `SignoViewer viewer` | The control element with the loaded document. |
| `DocumentDisplayDTO documentDisplay` | The display settings. |

| Return value | Description |
|---|---|
| `LoadImageFuture` | The result of the asynchronous processing. |

Usage:

```
SignoPdfSignerSTPad signer = new SignoPdfSignerSTPad("input.pdf",padApi);
// create the display configuration with all pages
DocumentDisplayDTO doc = signer.getDocumentDisplay(viewer);
// display the document on the signature pad
LoadImageFuture f = signer.displayDocument(viewer, doc);
// scroll to another position
padApi.setScrollPosition(0, x);
// wait until the image is uploaded for the new scroll position
f.waitForDisplayImage();
```

## 11.8 updateSignature method

This method updates a signature field on the signature device. It renders the page section of the field from the document again and loads it into foreground memory of the signature device.

```
public void updateSignature(SignoViewer viewer, DocumentDisplayDTO document,
     SignatureFieldInfoDTO field) throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `SignoViewer viewer` | The control element used to render the page section again. |
| `DocumentDisplayDTO document` | The display settings used to determine the position of the field. |
| `SignatureFieldInfoDTO field` | The field that is updated. |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    SignoPdfSignerSTPad signer =
            new SignoPdfSignerSTPad("input.pdf" , padApi);
    // start signature process
    signer.initSignature(...);
    // end signature process and save the signature in the document
    signer.confirmSignature(...);
    // reload the document
    viewer.loadDocument(...);
    // the signature is always visible on the pad - remove it
    padApi.clearSignature();
    // update the signature field on the pad
    signer.updateSignature(viewer, document, field);
} catch (SignoSignerException e) {
    // error handling
}
```

## 11.9 highlightFields method

This method highlights signature fields on the signature device with a fill colour and a rectangular frame. The fill and frame are optional. This method should only be used after a

document has been loaded in the signature device with `displayDocument()`. The highlighting can be removed again by applying neither the frame nor fill colour, or by using `updateSignature()`.

```
public void highlightFields(SignoViewer viewer, DocumentDisplayDTO document,
     SignatureFieldInfoDTO[] fields, Color fillColor, Color borderColor)
     throws SignoSignerException
```

| Parameter | Description |
|---|---|
| `SignoViewer viewer` | The control element used to render the page section again. |
| `DocumentDisplayDTO document` | The display settings used to determined the position of the fields. |
| `SignatureFieldInfoDTO[] field` | The affected fields. |
| `Color fillColor` | The fill colour for the signature fields. A transparent colour with alpha channel should be used. The fill is disabled if `null` is used. |
| `Color borderColor` | The colour for the frame of the signature field. The frame is disabled if `null` is used. |
| **Return value** | **Description** |
| – | - |

Usage:

```
try {
    SignatureFieldInfoDTO[] fields =
            SignoPdfSigner.getSignatureFieldInfo("input.pdf", false);
    // highlight all fields of the document
    signer.highlightFields(viewer, document, fields, fillColor,
            borderColor);
    // remove the highlight from all fields
    signer.highlightFields(viewer, document, fields, null, null);
} catch (SignoSignerException e) {
    // error handling
}
```

## 12 BiometricDataDTO class

The `de.signotec.pdf.signer.dto.BiometricDataDTO` class is a data object that contains the biometric data of a signature. Objects of this class are returned by means of the `SignoPdfSigner.getBiometricData()` method. As this data is saved in the document in an encrypted format, the matching certificate with the private RSA key for the signature in question is required to decrypt the biometric data.
The table below lists the Getter methods for this class.

| Method | Return | Description |
|---|---|---|
| `getDocHashAlgorithm` | `String` | Name of the algorithm that was used to calculate the hash code for the document |
| `getDocHashValue` | `String` | Hash code that the document had prior to signature |
| `getDocRecalcedHash-Value` | `String` | Newly calculated hash code of the document. If this value does not correspond to the value returned by the `getDocHashValue` method, the document has been manipulated. |
| `getBioHashAlgorithm` | `String` | Name of the algorithm that was used to calculate the hash code for the biometric data |
| `getBioRecalcedHash-Value` | `String` | Recalculated hash code of the biometric data with which the RSA signature can be verified |
| `getRSASignature` | `String` | RSA signature which was calculated in the signature device via the document hash and/or the biometry hash |
| `getHashType` | `String` | Defines the hash code that was signed in the signature device |
| | | `"HASH1"`: Document hash |
| | | `"HASH2"`: Biometrics hash |
| | | `"COMBINATION"`: combination of both hash codes |
| `getRSAScheme` | `String` | RSA scheme with which the RSA signature was made |
| `getPadSigningCert` | `X509Certif icate` | Certificate that can be used to verify the RSA signature |
| `getKeyGen` | `String` | Defines the origin of the private key that was used for the RSA signature: |
| | | `"Internal"`: the key has been generated in the signature device and has never left it |
| | | `"External"`: the key has been generated outside the signature device and uploaded |
| | | `"Factory"`: the key has been generated during manufacture and has never left the signature device |
| `getTimeStamp` | `Date` | Time at which the signature was captured |
| `getMachine` | `String` | Name of the computer on which the signature was captured |
| `getMacaddress` | `String` | MAC address of the computer on which the signature was captured |
| `getPadid` | `String` | Serial number of the signature device with which the signature was captured |
| `getPadmodel` | `String` | Model name of the signature device with which the signature was captured |
| `getPadVersion` | `String` | Firmware version of the signature device with which the signature was captured |

| | | |
|---|---|---|
| getBioData | String | Biometric data of the signature in SignData format |

Usage:

```
try {
    BiometricDataDTO bio = SignoPdfSigner.getBiometricData(
            "test.pdf", "sig1", "privateBio.jks", "AlsoSecret");

    if (bio.getDocHashAlgorithm() != null)
        System.out.println("Algorithm:" + bio.getDocHashAlgorithm());
    if (bio.getDocHashValue() != null)
        System.out.println("Hashcode:" + bio.getDocHashValue());
    if (bio.getDocRecalcedHashValue() != null)
        System.out.println("Recalced:" + bio.getDocRecalcedHashValue());
    if (bio.getBioHashAlgorithm() != null)
        System.out.println("Algorithm:" + bio.getBioHashAlgorithm());
    if (bio.getBioRecalcedHashValue() != null)
        System.out.println("Recalced:" + bio.getBioRecalcedHashValue());
    if (bio.getRSASignature() != null)
        System.out.println("Signature:" + bio.getRSASignature());
    if (bio.getHashType() != null)
        System.out.println("Type:" + bio.getHashType());
    if (bio.getRSAScheme() != null)
        System.out.println("Scheme:" + bio.getRSAScheme());
    if (bio.getKeyGen() != null)
        System.out.println("Key:" + bio.getKeyGen());
    if (bio.getTimeStamp() != null)
        System.out.println("SignDate:" + bio.getTimeStamp());
    if (bio.getMachine() != null)
        System.out.println("Machine:" + bio.getMachine());
    if (bio.getMacaddress() != null)
        System.out.println("Mac:" + bio.getMacaddress());
    if (bio.getPadid() != null)
        System.out.println("Serialnumber:" + bio.getPadid());
    if (bio.getPadmodel() != null)
        System.out.println("Model:" + bio.getPadmodel());
    if (bio.getPadVersion() != null)
        System.out.println("Version:" + bio.getPadVersion());
    if (bio.getBioData() != null)
        System.out.println("biometric data:" + bio.getBioData());
} catch (SignoSignerException e) {
    // error handling
}
```

## 13 BiproSignaturMetaDataDTO class

The `de.signotec.pdf.signer.dto.BiproSignaturMetaDataDTO` class is a data object that contains a signature's metadata defined according to the BiPRO standard. Objects of this class are returned by means of the `SignoPdfSigner.getBiproSignaturMetaData` method.
The table below lists the Getter methods for this class.

| Method | Return | Description |
|---|---|---|
| Signature metadata according to BiPRO standard | | |
| `getBiproSignatur-SoftwareHersteller` | `String` | Name of signature software manufacturer |
| `getBiproSignatur-SoftwareHersteller-Kontakt` | `String` | Contact at the signature software manufacturer |
| `getBiproSignatur-SoftwareVersion` | `String` | Version of the signature software |
| `getBiproSignatur-DeviceHersteller` | `String` | Name of signature device manufacturer |
| `getBiproSignatur-DeviceHersteller-Kontakt` | `String` | Contact at the signature device manufacturer |
| `getBiproSignatur-DeviceId` | `String` | ID of the signature device |
| `getBiproSignatur-DeviceModel` | `String` | Model name of the signature device |
| `getBiproSignatur-DeviceVersion` | `String` | Signature device version |
| `getBiproBioCertRef` | `String` | ID of the certificate that was used to encrypt the biometric data |
| `getBiproVersion` | `String` | Version of BiPRO standard |
| `getBiproPreSigning-ConfirmationMessage` | `String` | Text for confirmation dialog. If a text is set, it must be displayed before signing and confirmed by the user. Default is `null`. |
| Further, signotec-specific metadata | | |
| `getSignoUserid` | `String` | Name of the registered user |
| `getSignoMachine` | `String` | Name of the PC on which the document was signed |
| `getSignoMacaddress` | `String` | MAC address of the PC on which the document was signed |

## 14 CertInfoDTO class

The `de.signotec.pdf.signer.dto.CertInfoDTO` class is a data object that is included in the `SignatureFieldInfoDTO` class.
The table below lists the Getter methods for this class.

| Method | Return | Description |
|---|---|---|
| `getIssuer` | `String` | Name from the certificate |
| `getPublicKeySize` | `int` | Length of the public key in bit |
| `getValidFrom` | `Date` | Date from which the certificate is valid |
| `getValidTo` | `Date` | Date to which the certificate is valid |

## 15 SignatureFieldInfoDTO class

The `de.signotec.pdf.signer.dto.SignatureFieldInfoDTO` class is a data object that is returned as an array by means of the `SignoPdfSigner.getSignatureFieldInfo` method. The table below lists the Getter methods for this class.

| Method | Return | Description |
|---|---|---|
| `getFieldName` | `String` | Name of the signature field |
| `getTabOrder` | `int` | Tab order of the signature field |
| `getFieldOrder` | `int` | Field order of the signature field |
| `getName` | `String` | Name of the signer |
| `getReason` | `String` | Reason for the signature |
| `getLocation` | `String` | Location of the signature |
| `getContactInfo` | `String` | Contact information of the signer |
| `getTime` | `Date` | Time of signature |
| `getPage` | `int` | Page number on which the signature is located |
| `isRequired` | `boolean` | Mandatory field yes/no |
| `getFilter` | `String` | Signature handler |
| `getSubFilter` | `String` | Specification of the signature method |
| `getStatus` | `Field-Status` | Status of the signature (see `FieldStatus` enum) |
| `getBioStatus` | `Field-Status` | Status of the signature for the biometric data (see Enum FieldStatus) |
| `isCertExpired` | `boolean` | Certificate expired yes/no |
| `getRect` | `Rectangle-DTO` | Position specification of the signature field |
| `getCertInfo` | `CertInfo-DTO` | Certificate information (see Class CertInfoDTO) |
| `isBioCertExpired` | `boolean` | Certificate of the signature for the biometric data expired yes/no |
| `getBioCertInfo` | `CertInfo-DTO` | Certificate information of the signature for the biometric data (see CertInfoDTO class) |
| `getReasons` | `List <String>` | List of reasons from which a reason can be chosen when signing. The selected reason is set with `setReason()`.<br>See also `isReasonRequired()` and `SignatureFieldDTO.getReasons()`.<br>Default: `null` |
| `isReasonRequired` | `boolean` | Determines if the reason for signing is checked.<br>`true`: There must be a reason from the list of reasons when signing. If the list is not `null` and empty or contains a single reason ".", no reason can be given.<br>`false`: The reason is not checked when signing. Any text can be specified. |
| | | |

## 16 SigningDTO class

The `de.signotec.pdf.signer.dto.SigningDTO` class is a data object that contains information about the signature.
The following table lists the Getter and Setter methods for this class.

| Method | Value | Description |
|---|---|---|
| `getReason`<br>`setReason` | `String` | Reason for the signature |
| `isShowReason`<br>`setShowReason` | `boolean` | Establishes whether the reason is output visually in the signed signature field (otherwise contained only in the document metadata) |
| `getLocation`<br>`setLocation` | `String` | Location of the signature |
| `isShowLocation`<br>`setShowLocation` | `boolean` | Establishes whether the location is output visually in the signed signature field (otherwise contained only in the document metadata) |
| `getTimestampFormat`<br>`setTimestampFormat` | `SimpleData Format` | Specifies the format of the timestamp on the signature graphic; no timestamp is displayed for `null` |
| `getTimestampFont-Data`<br>`setTimestampFont-Data` | `byte[]` | The character set that is embedded in the document and is used to display the timestamp; if no character set is specified, then Helvetica is used but will not be embedded; this may result in the signed document not being PDF/A-compliant |
| `getRectangle`<br>`setRectangle` | `Rectangle-DTO` | Position specification of the signature field (is only required if the signature field is not yet available) |
| `getMaximize-Signature`<br>`setMaximize-Signature` | `boolean` | Specifies whether the signature is to be scaled to the signature field; this may only be `false` if `rectangle` has been set; the signature is scaled such that the rectangle of the entire LCD fits in the transferred rectangle, and the signature field is generated in precisely the same size as the signature |
| `getSignatureAlign`<br>`setSignatureAlign` | `Signature-Align` | Specifies how the signature in the signature field should be aligned in case the aspect ratio does not match the aspect ratio of the signature field |
| `getSignatureWidth`<br>`setSignatureWidth` | `int` | Specifies the width of the signature line (1 – 20) |
| `getSignatureColor`<br>`setSignatureColor` | `Color` | Specifies the colour of the signature line |
| `getPage`<br>`setPage` | `int` | Page specification of the signature field (is only required if the signature field is not yet available) |
| `getSubFilter`<br>`setSubFilter` | `SubFilter` | The value used for the /SubFilter element in PDF signing. |
| `getHashAlgorithm`<br>`setHashAlgorithm` | `HashAlgori thm` | The hash algorithm used to create the signature. The default is `SHA_256`.<br>The `SHA_256` algorithm updates the PDF version of the document to PDF 1.6 (Acrobat 7). |

## 17 RectangleDTO class

This class is included both in the `signopdf-signer` library and in the `signopdf-utilities` library as `de.signotec.pdf.signer.dto.RectangleDTO` and/or `de.signotec.pdf.utilities.dto.RectangleDTO` and is a data object that represents a rectangle in PDF coordinates. The values are indicated in pixels, the coordinate origin is on the left at the bottom.
The class contains the following constructors and methods:

| Method | Description |
|---|---|
| `RectangleDTO()` | Creates an empty rectangle with the origin at (0;0). |
| `RectangleDTO( float left, float right, float top, float bottom)` | Creates a rectangle with the specified coordinates. |
| `RectangleDTO( RectangleDTO other)` | Creates a copy of a rectangle from the same package. |
| `float getLeft() setLeft(float left)` | Getter and setter for the horizontal position of the left edge of the rectangle. |
| `float getRight() setRight(float right)` | Getter and setter for the horizontal position of the right edge of the rectangle. |
| `float getTop() setTop(float top)` | Getter and setter for the vertical position of the top edge of the rectangle. |
| `float getBottom() setBottom(float bottom)` | Getter and setter for the vertical position of the bottom edge of the rectangle. |
| `float getWidth()` | Returns the absolute width of the rectangle. |
| `float getHeight()` | Returns the absolute height of the rectangle. |
| `Rectangle toRectangle()` | Converts the rectangle to an AWT rectangle. The values are rounded to the nearest integer. |
| `RectangleDTO valueOf( RectangleDTO other)` | Converts a rectangle from the package `de.signotec.pdf.signer.dto` to a rectangle from the package `de.signotec.pdf.utilities.dto`. |

## 18 Enum FieldStatus

The `de.signotec.pdf.signer.enums.FieldStatus` enum provides information on the status of a signature. It is used by the `SignatureFieldInfoDTO` class.

| Value | Description |
|---|---|
| `VALID` | The signature is valid |
| `CHANGEDAFTERSIGNING` | The signature is valid, but subsequent modifications (further signatures) were conducted |
| `INVALID` | The document and/or the biometric data were manipulated |
| `EMPTY` | The signature field is not signed |
| `UNKNOWN` | The status could not be identified |

## 19 SignoPdfUtils class

Another main component is the `de.signotec.pdf.utilities.SignoPdfUtils` class. This class is a component for editing PDF documents. It has replaced the obsolete `de.signotec.pdf.utilities.SignoPdfUtilities` class since version 2.9.7.

### 19.1 addSignatureField method

This method adds a signature field to the copy of an existing document.

```
public static void addSignatureField(InputStream inDocument,
     OutputStream outDocument, SignatureFieldDTO field)
     throws SignoPdfUtilitiesException

public static void addSignatureField(File inDocument, File outDocument,
     SignatureFieldDTO field)
     throws SignoPdfUtilitiesException
```

| Parameter | Description |
|---|---|
| `InputStream / File inDocument` | `InputStream` or file for the document to which a field is to be added |
| `OutputStream / File outDocument` | `OutputStream` or file for saving the document to which a field has been added |
| `SignatureFieldDTO field` | SignatureFieldDTO object that contains the necessary information on the signature field |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
     SignatureFieldDTO field = new SignatureFieldDTO();
     field.setName("SignatureField");
     field.setRectangle(new RectangleDTO(50, 150, 300, 250));
     field.setPage(1);

     File input = new File("input.pdf");
     File output = new File("output.pdf");

     SignoPdfUtils.addSignatureField(input, output, field);
} catch (SignoPdfUtilitiesException e) {
       // error handling
}
```

### 19.2 searchText method

This method searches for the text indicated in a document.

```
public static List<SignatureFieldDTO> searchText(InputStream inDocument,
     String searchString, boolean caseSensitive, boolean wholeWords)
     throws InvalidParameterException, SignoPdfUtilitiesException

public static List<SignatureFieldDTO> searchText(InputStream inDocument,
     String searchArray[], boolean caseSensitive, boolean wholeWords)
     throws InvalidParameterException, SignoPdfUtilitiesException
```

```
public static List<SignatureFieldDTO> searchText(File inDocument,
     String searchString, boolean caseSensitive, boolean wholeWords)
     throws InvalidParameterException, SignoPdfUtilitiesException

public static List<SignatureFieldDTO> searchText(File inDocument,
     String searchArray[], boolean caseSensitive, boolean wholeWords)
     throws InvalidParameterException, SignoPdfUtilitiesException
```

| Parameter | Description |
|---|---|
| `InputStream / File inDocument` | `InputStream` or file for the document in which to search |
| `String searchString` | (Optional) Text to be searched |
| `String searchArray[]` | (Optional) Array of texts to be searched |
| `boolean caseSensitive` | Specifies whether case sensitivity should be applied in the search |
| `boolean wholeWords` | Specifies whether only whole words should be searched |
| **Return value** | **Description** |
| `List<SignatureFieldDTO>` | Found items |

Usage:

```
File pdfFile = new File("input.pdf");
List<SignatureFieldDTO> searchResult =
        SignoPdfUtils.searchText(pdfFile, "Signature", false, false);
```

## 19.3 addImage method

This method adds an image to the copy of an existing document.

```
public static void addImage(File inDocument, File outDocument, int page,
     BufferedImage image, int x, int y, int width, int height)
     throws SignoPdfUtilitiesException

public static void addImage(File inDocument, File outDocument, int page,
     byte[] imageData, int x, int y, int width, int height)
     throws SignoPdfUtilitiesException

public static void addImage(InputStream inDocument, OutputStream outDocument,
     int page, BufferedImage image, int x, int y, int width, int height)
     throws SignoPdfUtilitiesException

public static void addImage(InputStream inDocument, OutputStream outDocument,
     int page, byte[] imageData, int x, int y, int width, int height)
     throws SignoPdfUtilitiesException
```

| Parameter | Description |
|---|---|
| `InputStream / File inDocument` | `InputStream` or file for the document to which an image is to be added. |
| `OutputStream / File outDocument` | `OutputStream` or file for saving the document to which an image has been added. |
| `int page` | The page index. The first page is index 1. |
| `BufferedImage / byte[] image` | The image or the raw image data. |
| `int x` | The horizontal position of the image in pixels. The origin of the coordinate system is at the bottom left. |
| `int y` | The vertical position of the image in pixels. The origin of the coordinate system is at the bottom left. |
| `int width` | The display width in pixels. The image is stretched or compressed to that size. |

| | |
|---|---|
| `int height` | The displayed height in pixels. The image is stretched or compressed to that size. |
| **Return value** | **Description** |
| - | - |

Usage:

```
// add a signature image
try {
    SigPadApi padApi = ...;
    BufferedImage image = padApi.saveSignatureAsStream(300, 1, true,
            BufferedImage.TYPE_3BYTE_BGR);
    File sourceFile = new File("input.pdf");
    File targetFile = new File("output.pdf");

    SignoPdfUtils.addImage(sourceFile, targetFile, 1, image, 50, 75,
            image.getWidth(), image.getHeight());
} catch (Exception e) {
    // error handling
}
```

## 19.4 flattenFormFields method

This method can flatten a list of form fields, or all fields in a document. If the document does not contain at least one field name, a `SignoPdfUtilitiesException` is thrown.

```
public static void flattenFormFields(File inDocument, File outDocument,
     String… fieldNames)
     throws SignoPdfUtilitiesException

public static void flattenFormFields(InputStream inDocument,
     OutputStream outDocument, String… fieldNames)
     throws SignoPdfUtilitiesException
```

| Parameter | Description |
|---|---|
| `InputStream / File inDocument` | `InputStream` or file of the source document. |
| `OutputStream / File outDocument` | `OutputStream` or file of the target document. |
| `String… fieldNames` | List of field names to be flattened. An empty list selects all fields in the source document. |
| **Return value** | **Description** |
| - | - |

Usage:

```
// flatten signature fields with name “Signature1” and “Signature2”
try {
    File sourceFile = new File("input.pdf");
    File targetFile = new File("output.pdf");

    SignoPdfUtils.flattenFormField(sourceFile, targetFile,
            "Signature1", "Signature2");
} catch (SignoPdfUtilitiesException e) {
    // error handling
}
```

## 20 SignatureFieldDTO class

The `de.signotec.pdf.utilities.dto.SignatureFieldDTO` class is a data object that is required by the `SignoPdfUtilities.addSignatureField` method.
The following table lists the Getter and Setter methods for this class.

| Method | Return | Description |
|---|---|---|
| `getName`<br>`setName` | `String` | Name of the signature field |
| `getPage`<br>`setPage` | `int` | Page number on which the signature field should be added |
| `isRequired`<br>`setRequired` | `boolean` | Mandatory field yes/no (no is default) |
| `getRect`<br>`setRect` | `RectangleDTO` | Position specification of the signature field |
| `getReasons`<br>`setReasons` | `List<String>` | List of reasons from which a reason can be chosen when signing. See also `isReasonRequired()` and `SignatureFieldInfoDTO.getReasons()`.<br>Default: `null` |
| `isReasonRequired`<br>`setReasonRequired` | `boolean` | Determines if the reason for signing is checked.<br>`true`: There must be a reason from the list of reasons when signing. If the list is not `null` and empty or contains a single reason ".", no reason can be given.<br>`false`: The reason is not checked when signing. Any text can be specified. |
| `getBiproValues` | `SignatureFieldBiproDTO` | BiPRO values for empty signature fields. |

## 21 SubFilter enum

The `de.signotec.pdf.signer.enums.SubFilter` enum is used to establish the value for the /SubFilter element in PDF signing. This enumeration is used as an attribute in the `SigningDTO` class. The default value is `ADBE_PKCS7_DETACHED`.

| Value | Description |
|---|---|
| `ADBE_PKCS7_DETACHED` | Der Wert `adbe.pkcs7.detached` |
| `ADBE_X509_RSA_SHA1` | Der Wert `adbe.x509.rsa_sha1` |

## 22 HashAlgorithm enum

The `de.signotec.pdf.signer.enums.HashAlgorithm` enum is used to establish the hash algorithm for the signature. This enumeration is used as an attribute in the `SigningDTO` class. The default value is `SHA_256`.
If a `SHA_256` hash is used, the PDF version of the saved document is updated to PDF 1.6 (Acrobat 7).

| Value | Description |
|---|---|
| `SHA_1` | Die Signatur wird mit einem SHA-1 Hash erstellt. |
| `SHA_256` | Die Signatur wird mit einem SHA-256 Hash erstellt. |

## 23 PageDisplayDTO class

The `de.signotec.pdf.signer.dto.PageDisplayDTO` class is a data object that is used by the `de.signotec.pdf.signer.dto.DocumentDisplayDTO` class. It provides information for displaying a page on the signature device.
As a rule, all instances of this class are created and initialised by the API when calling `SignoPdfSignerSTPad.getDocumentDisplay()`.

The table below lists the properties of this class.

| Method | Return | Description |
|---|---|---|
| `getNumber`<br>`setNumber` | `int` | The page number in the original document. Required as a reference to the page when an arbitrary combination of pages is displayed. The number of the first page is 1.<br>*The value is administered automatically by the API.* |
| `getScale`<br>`setScale` | `double` | The scaling factor from the original size in the document to the size displayed on the signature device.<br>*The value is administered automatically by the API.* |
| `getBounds`<br>`setBounds` | `RectangleDTO` | The position and size of the page in the image memory of the signature device in pixels. Changing the page spacing with `DocumentDisplayDTO.setPageSpacing()` has an immediate effect on this rectangle.<br>*The value is administered automatically by the API.* |
| `getSignaturFields`<br>`setSignaturFields` | `List <SignatureFieldInfoDTO>` | The signature fields that are highlighted on the display. The position of the fields must be unchanged from the document. |

## 23.1 getPosition method

This method scales the size and position of a signature field from a document to the display size of the page.

```
public RectangleDTO getPosition(SignatureFieldInfoDTO field)
```

| Parameter | Description |
|---|---|
| `SignatureFieldInfoDTO field` | Any signature field that does not have to be contained in `getSignatureFields()`. The position and size is relative to the page origin and refers to the size in the document. |
| **Return value** | **Description** |
| `RectangleDTO` | Returns the scaled position and size for display on the signature device in pixels. The position is relative to the page origin. |

Usage:

```
try {
    PageDisplayDTO page = …;
    // read the fields with the original bounds in the document
    SignatureFieldInfoDTO[] fields =
            SignoPdfSigner.getSignatureFieldInfo("input.pdf", false);
    // transform its bounds to the image memory of the device
    RectangleDTO padPosition = page.getPosition(fields[0]);
    // do something with the result
```

```
    System.out.println(padPosition);
} catch (SignoSignerException e) {
    // error handling
}
```

## 24 DocumentDisplayDTO class

This class is a data object created and used by the `de.signotec.pdf.signer.SignoPdfSignerSTPad` class. If offers settings for the display of a document on the signature device.
As a rule, all instances of this class are created and initialised by the API when calling `SignoPdfSignerSTPad.getDocumentDisplay()`. Signature fields that are highlighted with a frame and fill colour on the display can be subsequently established with `addSignatureField()`.
If a toolbar from overlay memory is to be shown on the display, the scrolling free area behind the toolbar can be established with `setMarginTop()` and/or `setMarginBottom()`.

The table below lists the properties of this class.

| Method | Return | Description |
|---|---|---|
| `getFieldHighlightColor`<br>`setFieldHighlightColor` | `Color` | The fill colour for the signature fields. A transparent colour with alpha channel should be used. The fill is disabled when the value is set to `null`.<br>Default value: `DocumentDisplayDTO.FIELD_HIGHLIGHT_FILL_COLOR` |
| `getFieldHighlightBorderColor`<br>`setFieldHighlightBorderColor` | `Color` | The colour for the frame of the signature field. The frame is disabled when the value is set to `null`.<br>Default value: `DocumentDisplayDTO.FIELD_HIGHLIGHT_BORDER_COLOR` |
| `getMarginTop`<br>`setMarginTop` | `int` | The free border of scrollable image memory above the first page in pixels. A change affects the position (`PageDisplayDTO.getBounds()`) of the pages.<br>Default value: 0 |
| `getMarginBottom`<br>`setMarginBottom` | `int` | The free border of scrollable image memory below the last page in pixels.<br>Default value: 0 |
| `getBackgroundColor`<br>`setBackgroundColor` | `Color` | The colour used to fill the area behind and between the pages of the document. |
| `getPageSpacing`<br>`setPageSpacing` | `int` | The vertical distance (in pixels) between the pages. Changing it directly affects the position (`PageDisplayDTO.getBounds()`) of the pages. |
| `getPages`<br>`setPages` | `List<PageDisplayDTO>` | The pages to be displayed on the signature device. Do not use a page in different displays! The default value is `null`. |
| `getPage` | `PageDisplayDTO` | Returns the page with a certain number. The number is the position in the original document beginning with 1. (`PageDisplayDTO.getNumber()`) |
| `getPageCount` | `int` | Returns the number of pages. |

## 24.1 addSignatureField method

This method adds any number of fields to the highlighted signature fields. Only the fields of the pages already contained (`getPages()`) are considered.

```
public void addSignatureFields(SignatureFieldInfoDTO[] fields)
```

| Parameter | Description |
|---|---|
| `SignatureFieldInfoDTO[] fields` | The signature fields of the contained pages. The position and size is relative to the page origin and refers to the size in the document. |
| **Return value** | **Description** |
| - | - |

Usage:

```
try {
    // read the fields with the original bounds in the document
    SignatureFieldInfoDTO[] fields =
            SignoPdfSigner.getSignatureFieldInfo("input.pdf", false);
    // create the display configuration with all pages and add the fields
    SignoPdfSignerSTPad signer = new SignoPdfSignerSTPad("input.pdf",
            padApi);
    DocumentDisplayDTO doc = signer.getDocumentDisplay(viewer);
    doc.addSignatureFields(fields);
    // do something with the result
    signer.displayDocument(viewer, doc);
} catch (SignoSignerException e) {
    // error handling
}
```

## 24.2 getBounds method

This method returns the position and size of the displayed document in the image memory of the signature device. It does not include the margins. The top edge is always the top edge of the first page. The bottom edge is the bottom edge of the last page. The height is the sum of all pages including page separators. The rectangle and margins together form the required scrolling range on the signature device.

```
public RectangleDTO getBounds()
```

| Parameter | Description |
|---|---|
| - | - |
| **Return value** | **Description** |
| `RectangleDTO` | The section of the image memory needed to display the pages including the page separators, in pixels. |

Usage:

```
try {
    // read the fields with the original bounds in the document
    SignatureFieldInfoDTO[] fields =
            SignoPdfSigner.getSignatureFieldInfo("input.pdf", false);
    // create the display configuration with all pages and add the fields
    SignoPdfSignerSTPad signer = new SignoPdfSignerSTPad("input.pdf",
            padApi);
    DocumentDisplayDTO doc = signer.getDocumentDisplay(viewer);
    RectangleDTO docBounds = doc.getBounds();
    // do something with the result
```

```
    System.out.println(docBounds);
} catch (SignoSignerException e) {
    // error handling
}
```

## 24.3 getPosition method

This method converts the size and position of a signature field from a document to the absolute position and size in the image memory of the signature device.

```
public RectangleDTO getPosition(SignatureFieldInfoDTO field)
```

| Parameter | Description |
|---|---|
| `SignatureFieldInfoDTO field` | A signature field of a contained page. The position and size is relative to the page origin and refers to the size in the document. |
| **Return value** | **Description** |
| `RectangleDTO` | Returns the absolute position in the image memory and the size scaled for display on the signature device in pixels. |

Usage:

```
try {
    DocumentDisplayDTO doc = …;
    // read the fields with the original bounds in the document
    SignatureFieldInfoDTO[] fields =
SignoPdfSigner.getSignatureFieldInfo(
            "input.pdf", false);
    // transform its bounds to the image memory of the device
    RectangleDTO padPosition = doc.getPosition(fields[0]);
    // do something with the result
    System.out.println(padPosition);
} catch (SignoSignerException e) {
    // error handling
}
```

## 25 Klasse SignatureFieldBiproDTO

The class `de.signotec.pdf.utilities.dto.SignatureFieldBiproDTO` is a data object used by the class `SignatureFieldDTO` to create empty signature fields. It contains all supported BiPRO values that can be contained in an empty signature field. All values are also contained in class `BiproSignaturMetaDataDTO`.

The table below lists the properties of this class.

| Methode | Typ | Bedeutung |
|---|---|---|
| `getPreSigningConfirm-ationMessage`<br>`setPreSigningConfirm-ationMessage` | `String` | Confirmation dialog. If a text is set, it must be displayed before signing and confirmed by the user. (BiPRO 2.5) |