Dokumentation

signoPAD-API Java

Software-Komponenten zur Kommunikation mit
signotec Sigma, Zeta, Omega, Gamma, Delta und Alpha LCD Pads

Version: 4.1
Datum: 18.09.2020

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

Inhaltsverzeichnis

Impressum

1 Dokumentenhistorie

Version	Datum	Bearbeiter	Status/Bemerkung
3.12	16.08.2017	Markus Mensinger	Änderungen in Kapitel 8.1.3
3.13	10.11.2017	Markus Mensinger	Änderungen in Kapitel 8.3, 8.3.7.
3.14	30.11.2017	Markus Mensinger	Änderungen in Kapitel 3.1, 10.26-10.27, 10.35.2, 10.48-10.49, 10.56, 10.60-10.61, 10.66-10.67, 10.79-10.80, 10.83-10.84, 10.92 Kapitel 4.3, 10.93, 22 hinzugefügt.
3.15	09.01.2018	Markus Mensinger	Änderungen in Kapitel 3.1-3.4
3.16	24.01.2018	Markus Mensinger	Änderungen in Kapitel 3.6.
3.17	01.06.2018	Markus Mensinger	Änderungen in Kapitel 6.11, 8.1.3, 9.21, 10.60-10.61, 10.66, 10.90-10.91 Kapitel 6.10-6.13, 9.23-9.35, 10.94-10.97 hinzugefügt.
3.18	07.06.2018	Markus Mensinger	Änderungen in Kapitel 3.1
3.19	22.02.2019	Markus Mensinger	Änderungen in Kapitel 21
3.20	17.09.2019	Markus Mensinger	JWinPointer Bibliothek hinzugefügt Änderungen in Kapitel 3.1, 3.7, 8.3.1 Kapitel 3.8 und 6.14 hinzugefügt
3.21	18.11.2019	Markus Mensinger	Änderungen in Kapitel 10.7, 10.90
4.0	18.03.2020	Markus Mensinger	Schnittstellendokumentation in neue JavaDoc Dokumentation verschoben, Kapitel 4.1 und 4.2 eingefügt, Java 6 entfernt, Änderungen in Kapitel 3.1, 6
4.1	18.09.2020	Markus Mensinger	Zeta Pad in Kapitel 5.1.1, 5.2.1 eingefügt, Änderungen in Kapitel 6

2 Einleitung

Das signoPAD-API für Java enthält mehrere Komponenten, mit deren Hilfe ein Programmierer unterschiedlichste Funktionen rund um das Erfassen einer elektronischen Unterschrift in eigene Anwendungen integrieren kann. Das umfasst neben dem Erfassen der eigentlichen Unterschrift auch die Darstellung von Grafiken, Texten und Schaltflächen mit einem signotec LCD Pad oder einem beliebigen Pen Display.

Dieses Dokument bietet den Einstieg in die API, indem alle Informationen zu den technischen Möglichkeiten und Anforderungen beschrieben sind.

Schnittstellenbeschreibung

Die Schnittstellenbeschreibung für Programmierer enthält technische Informationen zu den einzelnen Java Klassen. Die Dokumentation im JavaDoc Format befindet sich im Ordner `doc/javadoc` des Auslieferungspakets.

```
doc/javadoc/index.html
```

3 Systemvoraussetzungen

Das signoPAD-API für Java ist grundsätzlich auf allen Windows-Versionen ab Windows 7 mit vollem Funktionsumfang lauffähig. Andere Betriebssysteme werden mit einem eingeschränkten Funktionsumfang unterstützt. Details finden Sie weiter unten.

3.1 JRE-Abhängigkeit

Das signoPAD-API Java benötigt die Java Runtime Environment (JRE) ab Version 1.7. Es werden sowohl die 32 Bit als auch die 64 Bit Version unterstützt.

Java 7

Wird die Anwendung in einem Umfeld ausgeführt, in dem das Code Signing Zertifikat geprüft wird, wird mindestens Java 7u76 benötigt. Ältere Java Versionen unterstützen nicht den Algorithmus der Signatur und weisen die Bibliothek als unsigniert zurück. Typische Anwendungsfälle, in denen nur signierter Code ausgeführt wird, sind Java Web Start Anwendungen und Applets.

Die folgende Tabelle enthält Internet Adressen unter denen Java kostenlos heruntergeladen werden kann.

Version	Download Adresse
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

Alle nicht standardmäßig in der JRE vorhandenen Komponenten werden mitgeliefert. In den folgenden Tabellen sind die Abhängigkeiten aufgeführt:

Abhängigkeit	Bemerkung
stpad-native.jar	Nur bei Verwendung der `SigPadFacade`.
bcprov-jdk15on-1.64.jar	-
jna-5.5.0.jar	-
jna-platform-5.5.0.jar	-
swt-gtk-linux-x86_64-3.8.2.jar swt-gtk-linux-x86-3.8.2.jar swt-win32-win32-x86_64-3.8.2.jar swt-win32-win32-x86-3.8.2.jar	Nur bei Verwendung des SWT Controls `SignatureWidget`
jpen-2.0.0.jar jpen-2-3.dll jpen-2-3-64.dll libjpen-2-4.so libjpen-2-4-x86_64.so	Nur bei Verwendung der `PenDisplayFacade` mit der Bibliothek `JPen`. Siehe auch Methode `PenDisplayFacade.getInstance()`.
jni4net.j-0.8.8.0.jar jwinpointer-se-1.0.0.jar	Nur bei Verwendung der `PenDisplayFacade` mit der Bibliothek `JWinPointer`. Siehe Methode `PenDisplayFacade.getInstance()`.

3.2 "Pure" Java-Schnittstelle

Die signotec LCD Signature Pads können (je nach Ausstattung) über TCP/IP verwendet werden. Für auf diese Art angeschlossene Geräte steht eine Schnittstelle mit eingeschränkter Funktionalität zur Verfügung, die keine plattformabhängige Bibliothek benötigt (`stpad-native.jar`) und somit auf allen Systemen verwendet werden kann, auf den Java-Anwendungen ausgeführt werden können. Details zu dieser Schnittstelle finden Sie im Kapitel „Einstiegspunkt SigPadPureFacade".

3.3 Windows

Unter Windows wird bei Verwendung der `SigPadFacade` die `stpad-native.jar` im Klassenpfad benötigt. Die darin enthaltene native Bibliothek wird automatisch mit JNA geladen.

Alternativ kann die .dll Datei aus der `stpad-native.jar` extrahiert werden und dessen Speicherort mit dem System.property `jna.library.path` beim Start der Anwendung festgelegt werden. Andere .dll- und .manifest-Dateien, die in den Version 8.0.22 und früher enthalten waren, werden nicht mehr benötigt.

3.4 Linux

Unter Linux wird bei Verwendung der `SigPadFacade` die `stpad-native.jar` im Klassenpfad benötigt. Die darin enthaltene native Bibliothek wird automatisch mit JNA geladen.

Alternativ kann die .so Datei aus der `stpad-native.jar` extrahiert werden und dessen Speicherort mit dem System.property `jna.library.path` beim Start der Anwendung festgelegt werden.

Bei Verwendung von HID/WinUSB-Geräten wird außerdem eine im System installierte libusb ab Version 1.0.16 benötigt, die unter http://www.libusb.org kostenlos heruntergeladen werden kann.

Ggf. müssen noch die Berechtigungen für libusb angepasst werden. Hierzu muss der MODE für USB auf 0666 geändert werden:

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

Anschließend muss `udev` neugestartet werden:

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

In Debian befindet sich diese Einstellung in der Datei `/lib/udev/rules.d/50-udev-default.rules` bzw. `/lib64/udev/rules.d/50-udev-default.rules`. Bei anderen Distributionen kann sich die Einstellung an einem anderen Ort befinden.

3.5 Java Advanced Imaging

Unterstützte Bildformate sind JPEG, BMP, WBMP und PNG. Um Unterschriften in anderen Bildformaten (JPEG 2000, PNM, RAW oder TIFF) exportieren zu können werden die Java Advanced Imaging Image I/O Tools benötigt. Diese können unter folgender Adresse heruntergeladen werden: https://jai-imageio.dev.java.net/

3.6 Security Policies

Die `SigPadPureFacade` sowie die Methode `SigPadApi.decryptSignatureData()` verwenden Kryptographieverfahren, die in Oracle Java bis Version 9 standardmäßig deaktiviert sind.

Bis Java 8u150 müssen die Dateien `local_policy.jar` und `US_export_policy.jar` im Ordner `%JRE_HOME%/lib/security` ausgetauscht werden. Sie finden die Dateien für die unterschiedlichen Java-Versionen im mitgelieferten `policy` Verzeichnis.

Ab Java 8u151 liegen die benötigten Policy Dateien der Java Installation bei und müssen mit dem Security-Property `crypto.policy=unlimited` ausgewählt werden *bevor* das JCE Framework initialisiert wird. Die signoPAD-API Java setzt dieses Property automatisch so früh wie möglich. Wenn aber die umschließende Anwendung das JCE Framework verwendet und initialisiert, muss es sicherstellen, dass das Security-Property gesetzt ist:

```
java.security.Security.setProperty("crypto.policy", "unlimited");
```

Weitere Informationen finden Sie in den Java 8u151 Release Notes.

3.7 JPen Bibliothek

Bei Verwendung der `PenDisplayFacade` kann die Bibliothek JPen zur Kommunikation mit dem Gerät verwendet. JPen Projektseite: https://sourceforge.net/projects/jpen

JPen lädt abhängig vom System eine native Bibliothek mit JNI. (siehe auch `JRE-Abhängigkeit`). Der Ordner, aus dem JNI die nativen Bibliotheken lädt, kann mit dem System.property `java.library.path` beim Start der Anwendung festgelegt werden.

Beispiel für den relativen Unterordner ‚lib':

```
> java -Djava.library.path=lib
```

Für die Erfassung mit dem Stift wird eine native Schnittstelle zum Pen Display benötigt, die i.d.R. in den Treibern des Herstellers enthalten ist. Unter Windows verwendet JPen bspw. die Wintab Schnittstelle. Für signotec Geräte sind alle benötigten Komponenten im Treiber „signotec Pen Display Manager" enthalten.

3.8 JWinPointer Bibliothek

Bei Verwendung der `PenDisplayFacade` kann die Bibliothek JWinPointer zur Kommunikation mit dem Gerät verwendet. JWinPointer funktioniert im Gegensatz zu `JPen` nur unter Microsoft Windows, benötigt .NET 4.5 oder höher und verwendet die Systemeigene API zur Erfassung der Daten.

JWinPointer benötigt Schreibrechte im Temp Ordner des Benutzers um dort native Bibliotheken abzulegen. Der Ordner wird standardmäßig der Umgebungsvariable `java.io.tmpdir` entnommen. Ein benutzerdefinierter Ordner kann mit der Variable `jwinpointer.library.path` festgelegt werden.

```
> java -Djwinpointer.library.path=C:\Temp
```

Für die Erfassung mit dem Stift wird eine native Schnittstelle zum Pen Display benötigt, die i.d.R. in den Treibern des Herstellers enthalten ist. Für signotec Geräte sind alle benötigten Komponenten im Treiber „signotec Pen Display Manager" enthalten.

4 Allgemeine Hinweise

4.1 Major Upgrades

Dieses Kapitel enthält wichtige Informationen zum Update zwischen Major Versionen der signoPAD-API Java. Es beschreibt die Änderungen, durch die ein Projekt nach dem Update der API nicht mehr kompilierbar ist.

Version 8.x auf 9.x

- Veraltete (`@Deprecated`) Methoden und Klassen entfernt. Entfernen Sie dessen Verwendung *vor* dem Upgrade der API.
- Minimale Java Version erhöht von JavaSE 6 auf JavaSE 7.
- Package `de.signotec.STPad` umbenannt in `de.signotec.stpad`
- Klasse `de.signotec.stpad.api.exceptions.Error` entfernt. Wird nicht mehr benötigt.
- Klasse `de.signotec.stpad.api.PenDisplayLibrary` verschoben nach `de.signotec.stpad.enums.PenDisplayLibrary`.
- Top-Level Exception `de.signotec.stpad.api.exceptions.SigPadApiException` durch neue `SigPadException` ersetzt.
- Methoden werfen jetzt auch Java Standard-Exceptions inkl. Unchecked Exceptions.
    - `java.lang.IllegalArgumentException` bei ungültigen Parametern
    - `java.lang.IllegalStateException` wenn eine Funktion im aktuellen Status nicht ausgeführt werden kann. Z.B. wenn die Unterschriften-erfassung gestartet wird, ohne vorher den Unterschriftenbereich festzulegen.
    - `java.lang.UnsupportedOperationException` wenn die Funktion von dem Signaturgerät oder der Fassade nicht unterstützt wird.
    - `java.io.IOException` bei I/O Fehlern.
    - `java.security.SignatureException` bei Verarbeitung von biometrischen Daten und Fehlern in Kryptographie Funktionen.
- Klassen `de.signotec.stpad.api.ForegroundBuffer`, `BackgroundBuffer` und `PermanentStore` entfernt. Die API verwendet nur noch dessen gemeinsame Superklasse `de.signotec.stpad.api.ImageMemory`. Der Speichertyp kann mit folgenden Methoden ermittelt werden:
    - `ImageMemory.isPermanentStore()`
    - `ImageMemory.isForegroundBuffer()`
    - `ImageMemory.isBackgroundBuffer()`
    - `ImageMemory.isOverlayBuffer()`
- Klassen `de.signotec.stpad.control.SignatureJPanel`, `SignatureCanvas` und `SignatureWidget` reagieren nicht mehr auf das `SigPadListener.errorOccurred()` Event. Der Fehler wird nicht mehr geloggt (`Throwable.printStackTrack()`) und dem Benutzer wird kein Fehlerdialog mehr angezeigt.
- Methode `de.signotec.stpad.api.SigPadApi.getSignatureData_Byte()` umbenannt in `getSignatureDataBytes()`.
- Methode `SignatureGraphics.setSampleRate(int)` entfernt. Wird nicht mehr benötigt.
- Konstante `de.signotec.stpad.enums.RSAScheme.NoOID` umbenannt in `NO_OID`.
- Abhängigkeit zu `commons-codec-1.4` Bibliothek entfernt.
- BouncyCastle auf Version 1.64 aktualisiert. Wenn es mit Java 7 zu JCE Validierungsfehlern beim Laden der Bibliothek kommt, sollte ein alternatives Jar verwendet werden.
  "Further Note (users of Oracle JVM 1.7 or earlier, users of "pre-Java 9" toolkits): As of 1.63 we have started including signed jars for "jdk15to18", if you run into issues with either signature validation in the JCE or the presence of the multi-release versions directory in the regular "jdk15on" jar files try the "jdk15to18" jars instead"
  Details siehe https://www.bouncycastle.org/latest_releases.html

4.2 Lizenzschlüssel

Die signoPAD-API Java wird im unregistriertem Zustand ausgeliefert. In diesem Zustand ist der Funktionsumfang während der Erfassung und Darstellung der Unterschrift auf einem Pen Display durch ein Wasserzeichen eingeschränkt. Bei Verwendung eines signotec Signature LCD Pad kann die API auch ohne Lizenzschlüssel *uneingeschränkt* verwendet werden.

Es existieren zwei unterschiedliche Arten von Lizenzschlüsseln:

1. Hardwaregebundene Einzelplatzlizenzen
2. Hardwareunabhängige Unternehmenslizenzen

Bei Lizenzen von **Typ 1** wird bei der Installation ein Autorisationsschlüssel (Software Code) generiert. Dieser ist an bestimmte Komponenten der Hardware gebunden und ist immer nur für einen Rechner gültig (Einzelplatz).

Mittels des mitgelieferten Programms `license-tool/license-tool.exe` kann nach Erwerb einer Einzelplatzlizenz ein Lizenzschlüssel beantragt und eingetragen werden.

Ist ein gültiger Lizenzschlüssel vorhanden, so wird dieser in der Registry des Computers hinterlegt und alle Komponenten der signoPAD-API Java sind damit vom Demostempel befreit.

Bei Lizenzen von **Typ 2** handelt es sich um hardwareunabhängige Unternehmenslizenzen, die nicht an einen Rechner/Arbeitsplatz gebunden sind und bei der der Schlüssel nicht in der Registry hinterlegt wird. Stattdessen muss der Lizenzschlüssel vor Verwendung der API durch den Aufruf der Methode `SigPadApi.setSerialKey()` gesetzt werden.

4.3 Datenformate

Unterschriften können als BMP, WBMP, JPEG, PNG, GIF, JPEG 2000, PNM, RAW und TIFF ausgegeben werden. In der Regel sollten Sie PNG verwenden, weil es die besten Ergebnisse bei kleinster Dateigröße bietet. JPEG ist ein Bildformat mit einer verlustbehafteten Komprimierung und wird nicht empfohlen.

4.4 SignData Strukturen

Die signoPAD-API Komponenten können eine erfasste Unterschrift als Datenstruktur im **SignData** Format zurückliefern. Es ist ein verschlüsseltes, komprimiertes, biometrisches Format, das in einer Datenbank und oder als Tag in einem TIFF-Dokument oder einem PDF-Dokument hinterlegt werden kann.

Für die (ISO-konforme) Signatur von PDF und TIFF-Dokumenten steht ein separates API (signoAPI) zur Verfügung. Dieses beinhaltet weitreichende Funktionen zum PDF-Management und vieles mehr. Bei Interesse fragen Sie bitte Ihren Kontakt bei signotec danach.

SignData-Strukturen können durch Implementierung des Interface `SigPadListener` aus dem Package `de.signotec.stpad.api.events` in Echtzeit visualisiert werden. Die Klassen `SignatureCanvas` (AWT), `SignatureJPanel` (Swing) und `SignatureWidget` (SWT) implementieren dieses Interface bereits und werden daher zur Visualisierung der Unterschriftenerfassung empfohlen.

4.5 Sicherheitskritische Daten

Passwörter und private Schlüssel gehören zu den sicherheitskritischen Daten und müssen mit besonderer Sorgfalt behandelt werden. Bei Verwendung dieser API sollten Sie neben den gängigen Sicherheitsstandards folgendes beachten, um das Sicherheitsniveau Ihrer Software möglichst hoch zu halten.

- Für Passwörter nur überschreibbare Datenstrukturen verwenden.
  Die signoPAD-API Java verwendet den Datentyp `char[]`. Die Daten eines unveränderlichen Typs wie `String` können nicht gezielt überschrieben bzw. gelöscht werden und verbleiben u.U. sehr lange im Arbeitsspeicher.
- Passwörter unmittelbar nach Verwendung löschen.
  Um das Zeitfenster für das Auslesen von Passwörtern aus dem Arbeitsspeicher so kurz wie möglich zu halten, sollten sie sofort nach Verwendung überschrieben werden. Die signoPAD-API Java bietet dafür die Methode `KeyLoader.clearPassword(char[])`.
- Private Schlüssel unmittelbar nach Verwendung löschen.
  Seit Java 8 sollten Schlüssel, die das `Destroyable` Interface implementieren, mit der Methode `destroy()` im Speicher überschrieben/unkenntlich gemacht werden sobald sie nicht mehr benötigt werden.

5 Bildspeicher

Die signotec LCD Signature Pads haben mehrere Bildspeicher, die von verschiedenen Methoden der Klasse `SigPadApi` angesprochen werden können. Ein Bildspeicher hat immer die Größe des Bildschirms und kann ein Bild in maximal dieser Größe speichern. Wird einem Speicher ein weiteres Bild hinzugefügt, überschreibt dieses die Bereiche, in denen es sich mit dem vorhandenen Speicherinhalt überlappt. Durch Speichern mehrerer Bilder in einem Speicher kann also eine Collage erstellt werden.

Je nach Modell steht eine unterschiedliche Anzahl an flüchtigen und nichtflüchtigen Speichern zur Verfügung.

5.1 Flüchtige Bildspeicher

Alle signotec LCD Signature Pads besitzen mind. zwei flüchtige Bildspeicher, einen, der den aktuellen Bildschirminhalt speichert (Vordergrundpuffer), und einen, in dem ein Bildinhalt zur Anzeige vorbereitet werden kann (Hintergrundpuffer). Grundsätzlich kann immer in beide Speicher geschrieben werden.

Der Inhalt der flüchtigen Bildspeicher geht beim Schließen der Verbindung zum Gerät verloren.

5.1.1 Modell Sigma und Zeta

Die beiden flüchtigen Bildspeicher haben jeweils die Größe des Bildschirms (Sigma 320x160 Pixel, Zeta 320x200 Pixel).

Die Übertragung und Darstellung von Bildern geht in der Regel so schnell, dass es zu keiner sichtbaren Verzögerung kommt. Bei komplexeren Darstellungen, die aus mehreren einzelnen Bildern bestehen, kann es aber sinnvoll sein, diese zuerst im Hintergrundpuffer zu speichern, um sie anschließend in den Vordergrundpuffer zu verschieben.

5.1.2 Modell Omega

Das Omega hat drei flüchtige Bildspeicher, zwei in der doppelten Größe des Bildschirms (640x960 Pixel) als Vorder- und Hintergrundpuffer sowie einen in Größe des Bildschirms (640x480 Pixel) als Overlay-Puffer, dessen Inhalt über den sonstigen Bildschirminhalt übergeblendet werden kann.

Die Geschwindigkeit des Bildaufbaus beim Modell Omega hängt von der Größe und dem Inhalt der Bilder ab, i. d. R. ist der Bildaufbau sichtbar. Daher sollten Bilder immer zunächst im Hintergrundpuffer gespeichert werden, um sie anschließend in den Vordergrundpuffer zu verschieben.

5.1.3 Model Gamma

Das Gamma hat drei flüchtige Bildspeicher, zwei, die größer sind als der Bildschirm (800x1440 Pixel), als Vorder- und Hintergrundpuffer sowie einen in Größe des Bildschirms (800x480 Pixel) als Overlay-Puffer, dessen Inhalt über den sonstigen Bildschirminhalt übergeblendet werden kann.

Ein Bild wird beim Modell Gamma immer erst angezeigt, nachdem es übertragen worden ist, der Bildaufbau ist nicht sichtbar. Die Geschwindigkeit der Bildübertragung hängt von der Größe und dem Inhalt der Bilder ab sowie von der Betriebsart. Nach Möglichkeit sollte das Gerät immer im WinUSB-Modus betrieben werden, hierzu ist ggf. die separate Installation des signotec WinUSB-Treibers erforderlich. Bei komplexeren Darstellungen, die aus mehreren einzelnen Bildern bestehen, ist es i. d. R. sinnvoll, diese zuerst im Hintergrundpuffer zu speichern, um sie anschließend in den Vordergrundpuffer zu verschieben.

5.1.4 Modell Alpha

Das Alpha hat drei flüchtige Bildspeicher, zwei, die größer sind als der Bildschirm (2048x2048 Pixel), als Vorder- und Hintergrundpuffer sowie einen in Größe des Bildschirms (768x1366 Pixel) als Overlay-Puffer, dessen Inhalt über den sonstigen Bildschirminhalt übergeblendet werden kann.

Ein Bild wird beim Modell Alpha immer erst angezeigt, nachdem es übertragen worden ist, der Bildaufbau ist nicht sichtbar. Die Geschwindigkeit der Bildübertragung hängt von der Größe und dem Inhalt der Bilder ab sowie von der Betriebsart. Nach Möglichkeit sollte das Gerät immer im WinUSB-Modus betrieben werden, hierzu ist ggf. die separate Installation des signotec WinUSB-Treibers nötig. Bei komplexeren Darstellungen, die aus mehreren einzelnen Bildern bestehen, ist es i. d. R. sinnvoll, diese zuerst im Hintergrundpuffer zu speichern, um sie anschließend in den Vordergrundpuffer zu verschieben.

5.1.5 Model Delta

Das Delta hat drei flüchtige Bildspeicher, zwei, die größer sind als der Bildschirm (1280x37600 Pixel), als Vorder- und Hintergrundpuffer sowie einen in Größe des Bildschirms (1280x800 Pixel) als Overlay-Puffer, dessen Inhalt über den sonstigen Bildschirminhalt übergeblendet werden kann.

Ein Bild wird beim Modell Delta immer erst angezeigt, nachdem es übertragen worden ist, der Bildaufbau ist nicht sichtbar. Die Geschwindigkeit der Bildübertragung hängt von der Größe und dem Inhalt der Bilder ab sowie von der Betriebsart. Nach Möglichkeit sollte das Gerät immer im WinUSB-Modus betrieben werden, hierzu ist ggf. die separate Installation des signotec WinUSB-Treibers erforderlich. Bei komplexeren Darstellungen, die aus mehreren einzelnen Bildern bestehen, ist es i. d. R. sinnvoll, diese zuerst im Hintergrundpuffer zu speichern, um sie anschließend in den Vordergrundpuffer zu verschieben.

5.1.6 Pen Display

Ein Pen Display hat keine physikalischen Bildspeicher. Für die Darstellung der grafischen Komponenten werden die zwei flüchtigen Bildspeicher im Arbeitsspeicher in der Größe des Erfassungsbereiches (`PenDisplayFacade.setDisplaySize()`) simuliert.

5.2 Nichtflüchtige Bildspeicher

Die signotec LCD Signature Pads besitzen je nach Modell eine unterschiedliche Anzahl an nichtflüchtigen Bildspeichern. Das Speichern von Bildern in nichtflüchtigen Bildspeichern dauert länger als das Speichern in flüchtigen Bildspeichern, dafür bleibt der Inhalt aber auch nach dem Ausschalten des Gerätes erhalten. Eine intelligente Speicherverwaltung erkennt, ob ein zu speicherndes Bild bereits im Gerät gespeichert ist, so dass es nur beim erstmaligen Speichern zu einer Verzögerung kommt.

5.2.1 Modell Sigma und Zeta

Die Modelle Sigma und Zeta besitzen einen nichtflüchtigen Bildspeicher in der Größe des Bildschirms (Sigma 320x160 Pixel, Zeta 320x200 Pixel), der nur für das Standby-Bild genutzt werden kann. Aufgrund der schnellen Übertragung und Darstellung von Bildern ist es nicht nötig, andere für den Programmablauf nötige Bilder dauerhaft speichern zu können.

5.2.2 Modell Omega

Das Modell Omega besitzt elf nichtflüchtige Bildspeicher, die für das Standby-Bild, die Diaschau und Optimierungen des Programmablaufs verwendet werden können. Die Bildspeicher, die für das Standby-Bild oder die Diaschau verwendet werden, sind schreibgeschützt und können nur durch das Deaktivieren des Standby-Bildes oder der Diaschau wieder freigegeben werden.

Ein nichtflüchtiger Bildspeicher hat die doppelte Größe des Bildschirms (640x960 Pixel), zehn Speicher haben die Größe des Bildschirms (640x480 Pixel).

Soll ein nichtflüchtiger Bildspeicher verwendet werden, muss dieser zunächst reserviert werden. Dies geschieht durch Aufruf der Methode `ImageMemory.requestPermanentStore()`. Die Größe eines Speichers kann über die Eigenschaften `ImageMemory.getWidth()` und `ImageMemory.getHeight()` abgefragt werden.

5.2.3 Model Gamma

Das Modell Gamma besitzt zehn nichtflüchtige Bildspeicher, die für das Standby-Bild, die Diaschau und Optimierungen des Programmablaufs verwendet werden können. Die Bildspeicher, die für das Standby-Bild oder die Diaschau verwendet werden, sind schreibgeschützt und können nur durch das Deaktivieren des Standby-Bildes oder der Diaschau wieder freigegeben werden.

Die zehn nichtflüchtigen Speicher haben die gleiche Größe wie der Bildschirm (800x480 Pixel).

Soll ein nichtflüchtiger Bildspeicher verwendet werden, muss dieser zunächst reserviert werden. Dies geschieht durch Aufruf der Methode `ImageMemory.requestPermanentStore()`. Die Größe eines Speichers kann über die Eigenschaften `ImageMemory.getWidth()` und `ImageMemory.getHeight()` abgefragt werden.

Im Gegensatz zum Omega ist die Verwendung der nichtflüchtigen Speicher beim Gamma zur Optimierung des Programmablaufs in der WinUSB-Betriebsart nicht nötig, weil die Bildübertragung sehr schnell ist. Das hängt aber immer vom Einzelfall und sollte individuell vom Entwickler entschieden werden.

5.2.4 Modell Alpha

Das Modell Alpha besitzt zehn nichtflüchtige Bildspeicher, die für das Standby-Bild, die Diaschau und Optimierungen des Programmablaufs verwendet werden können. Die Bildspeicher, die für das Standby-Bild oder die Diaschau verwendet werden, sind schreibgeschützt und können nur durch das Deaktivieren des Standby-Bildes oder der Diaschau wieder freigegeben werden.

Die zehn nichtflüchtigen Speicher haben die gleiche Größe wie flüchtigen Speicher (2048x2048 Pixel).

Soll ein nichtflüchtiger Bildspeicher verwendet werden, muss dieser zunächst reserviert werden. Dies geschieht durch Aufruf der Methode `ImageMemory.requestPermanentStore()`. Die Größe eines Speichers kann über die Eigenschaften `ImageMemory.getWidth()` und `ImageMemory.getHeight()` abgefragt werden.

Im Gegensatz zum Omega ist die Verwendung der nichtflüchtigen Speicher beim Alpha zur Optimierung des Programmablaufs in der WinUSB-Betriebsart nicht nötig, weil die Bildübertragung sehr schnell ist. Das hängt aber immer vom Einzelfall und sollte individuell vom Entwickler entschieden werden.

5.2.5 Model Delta

Das Modell Delta besitzt 32 nichtflüchtige Bildspeicher, die für das Standby-Bild, die Diaschau und Optimierungen des Programmablaufs verwendet werden können. Die Bildspeicher, die für das Standby-Bild oder die Diaschau verwendet werden, sind schreibgeschützt und können nur durch das Deaktivieren des Standby-Bildes oder der Diaschau wieder freigegeben werden.

Die 32 nichtflüchtigen Speicher haben die gleiche Größe wie der Bildschirm (1280x800 Pixel).

Soll ein nichtflüchtiger Bildspeicher verwendet werden, muss dieser zunächst reserviert werden. Dies geschieht durch Aufruf der Methode `ImageMemory.requestPermanentStore()`. Die Größe eines Speichers kann über die Eigenschaften `ImageMemory.getWidth()` und `ImageMemory.getHeight()` abgefragt werden.

Im Gegensatz zum Omega ist die Verwendung der nichtflüchtigen Speicher beim Delta zur Optimierung des Programmablaufs in der WinUSB-Betriebsart nicht nötig, weil die Bildübertragung sehr schnell ist. Das hängt aber immer vom Einzelfall und sollte individuell vom Entwickler entschieden werden.

5.2.6 Pen Display

Ein Pen Display hat keine nichtflüchtigen Bildspeicher.

5.3 Kopieren zwischen Bildspeichern

Zwischen den meisten verfügbaren Bildspeichern kann der Inhalt kopiert werden. Der Inhalt des Hintergrundpuffers kann nicht in den Vordergrundpuffer kopiert, sondern nur verschoben werden. Der Inhalt des Overlay-Puffers kann nicht kopiert, sondern nur über den Bildschirminhalt übergeblendet werden.

Typische Kopiervorgänge sind das Kopieren aus einem nichtflüchtigen Bildspeicher in einen flüchtigen Bildspeicher und das Kopieren aus dem flüchtigen Hintergrundpuffer in den Vordergrundpuffer. Das Kopieren eines Bildspeichers innerhalb des Gerätes ist immer schneller als das Senden dieses Bildes vom PC zum Gerät. Details zum Kopieren entnehmen Sie bitte der Beschreibung der Methoden `SigPadApi.setImageFromStore()` und `SigPadApi.setOverlayArea()`.

5.4 Der typische Ablauf

Die meisten Anwendungen verwenden für den Unterschriftvorgang immer die gleichen Bilder mit evtl. variablen Anteilen (z. B. dokumentenbezogene Texte). Daher ist es sinnvoll, Bilder, die bei jedem Ablauf gleich sind, nach Möglichkeit in einem der nichtflüchtigen Bildspeicher abzulegen. Nachfolgend der typische Ablauf für dieses Szenario:

Zunächst werden die Bilder geladen, die dauerhaft im Gerät gespeichert werden sollen, da sie sich selten verändern. Dafür wird die statische Methode `ImageMemory.requestPermanentStore()` verwendet. Die Methode gibt eine Referenz auf einen nichtflüchtigen Bildspeicher zurück, die dann verwendet werden kann, um Bilder oder Texte diesem Speicher hinzuzufügen. Steht kein nichtflüchtiger Bildspeicher zur Verfügung, gibt die Methode eine Referenz auf einen flüchtigen Speicher zurück, die verwendet werden kann, um Bilder oder Texte dem Hintergrundspeicher hinzuzufügen.

Um immer das ggf. aus mehreren Texten und Bildern zusammengesetzte Bild mit dem bereits im Gerät gespeicherten Bild vergleichen zu können, werden die Texte und die Bilder, die einem nichtflüchtigen Speicher hinzugefügt werden, zunächst nur lokal gespeichert und erst beim Aufruf von `SigPadApi.setImageFromStore()` oder `SigPadApi.configSlideShow()` tatsächlich an das Gerät gesendet. Erst beim Aufruf einer dieser Methoden kommt es damit ggf. zu einer merklichen Verzögerung.

```
ImageMemory store = ImageMemory.requestPermanetStore();
padApi.setImage(10, 10, img1, store);
padApi.setText(220, 160, SigPadAlign.LEFT, "Signature:", store);
padApi.setImage(220, 400, img2, store);
```

Die Inhalte können nun in einen der flüchtigen Bildspeicher kopiert werden, i. d. R. sollte das der Hintergrundpuffer sein. Sollten die Bilder bereits in den Hintergrundpuffer geschrieben worden sein, weil kein nichtflüchtiger Speicher zur Verfügung stand (s. o.), hat folgende Methode keinerlei Funktion, produziert aber auch keinen Fehler, kann also gefahrlos aufgerufen werden.

```
ImageMemory backBuffer = ImageMemory.requestBackgroundBuffer();
padApi.setImageFromStore(dest1, backBuffer);
```

Nun können dem Hintergrundpuffer Inhalte hinzugefügt werden, die sich bei jedem Unterschriftenvorgang verändern.

```
padApi.setImage(120, 400, img3, backBuffer);
padApi.setText(220, 180, SigPadAlign.LEFT, "01.01.2010", backBuffer);
```

Im Hintergrundpuffer befindet sich jetzt eine Collage aus zwei Bildern und einem Text, die aus einem nichtflüchtigen Bildspeicher kopiert worden sind, und einem Bild und einem Text, die vom PC gesendet worden sind. Diese Collage kann jetzt in den Vordergrundpuffer kopiert und somit auf dem Bildschirm angezeigt werden. Der gesamte Bildaufbau vorher ist im Hintergrund geschehen und somit „unsichtbar".

```
padApi.setImageFromStore(backBuffer);
```

Der beschriebene Ablauf muss nach jedem Öffnen einer Verbindung erneut durchgeführt werden. Beim Schließen einer Verbindung gehen alle Informationen über reservierte Speicher verloren. Nur die Information, welcher Bildinhalt in welchem nichtflüchtigen Speicher enthalten ist, bleibt im Gerät erhalten (auch wenn es stromlos gesetzt wird).

6 Einstiegspunkte

Für eine Unterschriftenerfassung sind die hier beschriebenen Einstiegspunkte möglich. Der Einstieg beginnt immer mit der Auswahl der Fassade abhängig davon welches Gerät, Technik oder Verbindungstyp verwendet werden soll. Anschließend wird mithilfe der Fassade ein `SigPadDevice` Objekt erstellt, was repräsentativ für ein angeschlossenes Signaturgerät steht. Zuletzt wird eine Instanz der `SigPadApi` Klasse für das `SigPadDevice` erstellt. Die `SigPadApi` Klasse ist die wichtigste Klasse wenn es um das Verwenden des Signaturgerätes geht.

Für Details siehe Schnittstellenbeschreibung.

6.1 SigPadFacade

Einer der drei Einstiegspunkte ist die Klasse `de.signotec.stpad.api.SigPadFacade`. Diese stellt allgemeine Methoden zur Initialisierung und Verwendung des signoPAD-API für Java zur Verfügung und sollte immer benutzt werden, sofern die plattformabhängigen Bibliotheken für Windows und Linux verwendet werden können.

Da es sich hierbei um eine Singleton-Klasse handelt, ist kein öffentlicher Konstruktor vorhanden, der Aufruf erfolgt über die `getInstance()` Methode. Im zweiten Schritt muss `initializeApi()` aufgerufen werden, damit wird die Schnittstelle initialisiert. Schritt 3 ist der Aufruf der `getSignatureDevices()` Methode, die alle angeschlossenen Pads als `SigPadDevice`-Objekte zurückliefert. Mit so einem Objekt kann dann die eigentliche Hauptfunktionalität in der Klasse `SigPadApi` aufgerufen werden. Nach Abschluss aller Operationen wird `finalizeApi()` aufgerufen, welches das signoPAD-API für Java ordnungsgemäß beendet. Anwendungen, die nur gelegentlich auf das Signaturgerät zugreifen müssen, können diesen Vorgang für jede Unterschrift vollständig durchführen, üblicherweise sollte das API aber beim Start der Anwendung initialisiert und beim Beenden der Anwendung erst finalisiert werden.

Im Folgenden wird ein schematischer Ablauf der Aufrufe gezeigt:

```
SigPadFacade facade = SigPadFacade.getInstance();
// initialize
facade.initializeApi();
// search for devices
SigPadDevice[] pads = facade.getSignatureDevices();
// create API for device
SigPadApi padApi = new SigPadApi(pads[0]);
// some custom operations
padApi.XXX();
// clean up
facade.finalizeApi();
```

6.2 SigPadPureFacade

Der zweite Einstiegspunkt ist die Klasse `de.signotec.stpad.api.SigPadPureFacade`. Diese stellt allgemeine Methoden zur Initialisierung und Verwendung des signoPAD-API für Java zur Verfügung und kann nur benutzt werden, wenn ein signotec LCD Signature Pad über Ethernet im LAN angeschlossen ist. Der Vorteil dieses Einstiegspunktes ist, dass keine plattformanhängigen Bibliotheken verwendet werden und somit keine Beschränkung auf Windows und Linux besteht.

Da es sich hierbei um eine Singleton-Klasse handelt, ist kein öffentlicher Konstruktor vorhanden, der Aufruf erfolgt über die `getInstance()` Methode. Im zweiten Schritt muss durch Aufruf der `getSignatureDevices()` Methode nach einem angeschlossenen Pad gesucht werden, das als `SigPadDevice` zurückgegeben wird. Mit diesem Objekt kann dann die eigentliche Hauptfunktionalität in der Klasse `SigPadApi` aufgerufen werden.

Im Folgenden wird ein schematischer Ablauf der Aufrufe gezeigt:

```
SigPadPureFacade facade = SigPadPureFacade.getInstance();
// search for devices
String address = "192.168.100.100";
int port = 1002;
SigPadDevice[] pads = facade.getSignatureDevices(address, port, false);
// create API for device
SigPadApi padApi = new SigPadApi(pads[0]);
// some custom operations
padApi.XXX();
```

6.3 PenDisplayFacade

Der dritte Einstiegspunkt ist die Klasse `de.signotec.stpad.api.PenDisplayFacade`. Sie bietet die Möglichkeit eine Unterschrift mit dem Stift auf einem Pen Display, z.B. dem signotec Delta PD, oder mit der Maus auf dem Desktop zu erfassen. Als Eingabefläche dient die grafische Swing Komponente `SignatureJPanel` oder `SignatureCanvas`.

Qualitativ unterscheidet sich die Unterschrift mit dem Stift stark von der Unterschrift mit der Maus. Die Genauigkeit der erfassten Punkte entspricht mit der Maus der Bildschirmauflösung. Beim Stift ist die Genauigkeit abhängig vom Digitizer i.d.R. um ein vielfaches höher. Die Maus bietet nur zwei Druckwerte (Taste gedrückt / Taste nicht gedrückt). Mit dem Stift sind über 1000 Druckwerte möglich.

Für die Erfassung mit dem Stift sind Treiber erforderlich. Siehe dazu Kapitel `JPen Bibliothek` und `JWinPointer Bibliothek`.

Die Verwendung der Klasse teilt sich in drei Schritte.

1. Erstellung eine neuen Instanz mit der Methode `getInstance()`.
2. Einstellen der Pen Display Eigenschaften wie Seriennummer, Größe und Auflösung des Erfassungsbereiches und ob Stift und/oder Maus aktiv ist.
3. Aufruf der `getSignatureDevice()` Methode, die eine `SigPadDevice` Instanz liefert, mit der die eigentliche Hauptfunktionalität in der Klasse `SigPadApi` aufgerufen wird.

Im Folgenden wird ein schematischer Ablauf der Aufrufe gezeigt:

```
SigPadApi.setSerialKey("myCompanyKey");
PenDisplayFacade facade = PenDisplayFacade.getInstance();
// set the serial number "1111" of the device
facade.setSerialNumber(1111L);
// use pen only, not the mouse
facade.setPenEnabled(true);
facade.setMouseEnabled(false);
// record the pen in area 320x240
facade.setDisplaySize(320, 240);
// create device
SigPadDevice device1 = facade.getSignatureDevice();
if (device1 != null) {
    SigPadApi padApi = new SigPadApi(device1);
    // open/connect device
    SignatureJPanel panel = new SignatureJPanel(padApi, 320, 240);
    padApi.openDevice(panel);
    // custom operations
    padApi.XXX();
}
```

Der Funktionsumfang der Fassade ist durch ein Wasserzeichen bei der Erfassung und Darstellung der Unterschrift eingeschränkt. Das Wasserzeichen kann durch Erwerb einer Lizenz von der signotec GmbH entfernt werden. Siehe Kapitel Lizenzschlüssel.

7 Schlüssel und Zertifikate

Die Klasse `de.signotec.stpad.api.util.KeyLoader` ist eine Hilfsklasse zum Laden von privaten Schlüsseln und Zertifikaten. Es bietet vereinfachten Zugriff auf Java Keystores, PKCS#12 formatierte Dateien und Zertifikatsdateien. Details siehe Schnittstellenbeschreibung.

8 Steuerelemente zur Visualisierung

Das Package `de.signotec.stpad.control` enthält visuelle Steuerelemente auf Basis von Swing, AWT oder SWT, zur Echtzeitdarstellung der erfassten Unterschriften.

Alle Steuerelemente können in zwei Modi betrieben werden. Zum einen zur Darstellung einer SignData Struktur (einer Unterschrift) und zum anderen zur Visualisierung der Unterschriftenerfassung.

8.1 Modus SignData

Der SignData-Modus ist aktiv, wenn ein Konstruktor ohne `SigPadApi` Parameter verwendet wird.

- Das Steuerelement ist nicht mit dem Pad verbunden und kann nicht zur Darstellung der Unterschriftenerfassung verwendet werden.
- Interaktive Steuerung von HotSpots und Scrollen mit der Maus sind nicht möglich.
- Das Steuerelement kann eine beliebige Größe erhalten.
- Die darzustellende Unterschrift wird mit `setSignatureData()` oder `animateSignature()` festgelegt.

8.2 Modus Unterschriftenerfassung

Der Unterschriftenerfassung-Modus ist aktiv, wenn ein Konstruktor mit `SigPadApi` Parameter verwendet wird.

- Das Steuerelement ist mit dem Pad verbunden und stellt das Pad-Display während der Unterschriftenerfassung dar. Dafür muss das Steuerelement über den `SigPadListener` mit der `SigPadApi` Klasse verbunden werden. (`SigPadApi.addSigPadListener(ctrl)`)
- Das Steuerelement hat eine feste Größe, die nicht geändert werden kann.
- Die Methoden `setSignatureData()` und `animateSignature()` können nicht verwendet werden.
- Interaktive Steuerung von HotSpots und das Scrollen mit der Maus ist nach Aktivierung mit `setMouseEnabled(true)` möglich.