EGIZ E-Government Innovationszentrum

www.egiz.gv.at
E-Mail: post@egiz.gv.at
Telefon: ++43 (316) 873 5514
Fax: ++43 (316) 873 5520
Inffeldgasse 16a / 8010 Graz / Austria

Signatur für elektronische Aktenführung

(PDF-Amtssignaturen)

Anwenderdokumentation

Version 2.7.1, 1.Juli 2007

Wilfried Lackner – wlackner@iicm.edu
Wolfgang Prinz – wprinz@iicm.edu

EGIZ-Kontakt: Thomas Rössler – thomas.roessler@egiz.gv.at

Inhaltsverzeichnis

Dokument-Historie

Datum	Version	Autor / Organisation	Änderungen
25.08.2006	0.9.0	Wilfried Lackner Wolfgang Prinz	Dokument erstellt.
20.09.2006	1.0.0 ENTWURF	Thomas Rössler (EGIZ)	Dokument formatiert, tw. korrigiert.
01.07.2007	2.7.1	Thomas Rössler (EGIZ)	Anpassung pos-Parameter.

EGIZ: PDF-Amtssignaturen (PDF-AS) Dokumentation

Grundsätzliches

Die PDF-AS Applikation ist als command line Tool und als Web Anwendung ausgelegt. Sämtliche Konfigurationen Pfade, Templates usw. sind voneinander unabhängig. D.h. sowohl das command line Tool, als auch die Web Anwendung kann unabhängig voneinander installiert werden.

Changes/Historie

Version 2.7.1

1. Anpassung pos-Parameter.

Version 2.7

1. Algorithmus AbsoluteTextsignatur geändert: Semantische Äquivalenz fällt weg, statt dessen Horizontale Breite berücksichtigen.
2. Änderungen bei der Absoluten Positionierung: Fußzeilen Positionierung beschrieben.

Version 2.6

1. Ausführungen über SIG_NAME erweitert.
2. Hinweis auf Statische Felder.
3. Hinweis auf die Unterschiede Text- Binärsignatur.
4. Anpassung des Algorithmus Absolute Textsignatur.

Version 2.5

1. Nähere Beschreibung der RSA/EDCA config Parameter.
2. Explizites Erwähnen der KeyboxIdentifiers und MOA Keys etc.
3. Definition der Required-Keys/Fields.
4. Nähere Beschreibung und Hinweis bezüglich SIG_NAME.

Version 2.4

5. Kleinere Änderungen wegen der absoluten Textsignatur.
6. Aufnahme der Algorithmen bezüglich „Absolute Textsignatur“ in die Entwickler Dokumentation.

Version 2.3

7. Konfigurationsdatei aufbereiten.
8. Dokumentation anwendergerecht aufbereiten.
9. Beschreibung der OIDs Verwaltungseigenschaft und Dienstleistereigenschaft in config.

Version 2.2

10. available_for_web und available_for_commandline Parameter in Konfiguration.
11. Seite -1 für “neue Seite anlegen” bei der absoluten Positionierung.

Version 2.1

1. Erweiterte Beschreibung der phlength Felder.

Version 2.0

1. Zahlreiche Änderungen am bisherigen Framework.
2. Einführung der normierten URN Kennzeichnungen in Signaturblöcken.
3. Mehrfachsignaturen
4. Textuelle und binäre Signaturen laut Spezifikation
5. Verwendung Incremental Update
6. Berücksichtigung von Bildern bei der Position des Signaturblocks
7. Absolute Positionierung
8. Anbindung an LDAP Interface (das explizite LDAP API muss noch von EGIZ geliefert werden)
9. Unterstützung der A1 Signatur (noch Kommunikation mit Telekom notwendig)
10. Entsprechende Anpassung der Dokumentation

Version 1.4

1. Kleinere Veränderungen

Version 1.3

1. Algorithmus zur Erkennung und Separation von Signaturblöcken wurde geändert.
2. In der HTML Voransicht werden beim Signaturblock nur die zur Rekonstruktion notwendigen Daten angezeigt.
3. Beim command line Tool wird die Angabe eines „Result“ – File sofort übernommen. Das „renaming“ aus Version 1.3 wurde ersetzt.
4. Beim Normalisieren wurde ein neues Leerzeichen gefunden: \u00a0 → non breaking space
5. Externe Fehlermeldungen werden an den Client weitergereicht → z.B. Verbindung zu MOA kann nicht aufgebaut werden, etc.

6. Anpassung der Dokumentation:
   Änderung der Nomenklatur: Signatur Block ➔ Sichtbare Signatur im PDF Dokument, Bildmarke ➔ Bild im Signatur Block, das Wort Signaturmarke wurde entfernt
   7.3 zusätzliches replace von non breaking spaces

Version 1.2

1. Signaturalgorithmus configurierbar:

```
cert.alg.rsa=http://www.w3.org/2000/09/xmldsig#rsa-sha1
cert.alg.ecdsa=http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1
```

2. Voransicht in der Webanwendung zeigt den extrahierten Rohtext an → Unterschied zu BKU SecureViewer, dort wird der normalisierte Dokument Text angezeigt
3. Dateien im Download Verzeichnis /dld sollten automatisch nach beenden der Web Anwendung gelöscht werden
4. Dateien in den temporärne Verzeichnissen sollten automatisch gelöscht werden, jedoch bleiben beim Signiervorgang trotzdem diverse temporäre Dateien bestehen. Diese müssen manuell gelöscht werden.
5. Mehrfachsignatur wird verhindert; → die Dokumentation wurde dahingehend nicht verändert. Wird ein signiertes Dokument nochmal dem Signier Request zugefügt, erfolgt ein Abbruch und eine Rückmeldung, dass das Dokument bereits signiert wurde. Die Mehrfachsignierung wird in einer weiteren Version umgesetzt.
6. Rückmeldung beim Prüfrequest von MOA, falscher Text, wurde behoben. Fehler lag in der Applikation, nicht in der Konfiguration.
7. Import von Zertifikaten: beim Import von Zertifikaten werden nur .cer codierte Dateien im Format base64 akzeptiert. Es erfolgt keine Konvertierung von .der codierten binären Formaten. Das von Windows akzeptierte base64 Format mit -----BEGIN CERTIFICATE-----base64Code-----END CERTIFICATE----- wird entsprechend normalisiert und übernommen. Binäre und andere Formate die dieser Konvention nicht entsprechen, werden zwar übernommen, führen aber beim Prüfrequest zu falschen Ergebnissen.
8. Voransicht im IE beim Prüfrequest ist jetzt stabil: rekonstruierte Bildmarke passt sich der Fenstergröße an → IE Bug wurde durch Javascript Code umgangen
9. Anpassung der Dokumentation:
   2.2.1: neue Einträge
   4: Zertifikate übernehmen
   6.4: temporäre Ordner
   6.4: Download Ordner
   8: Problembereiche erweitert

Konfiguration config.properties

Grundsätzliches

Die Konfigurationsdateien befinden sich in folgendem Verzeichnis:

```
PDF-AS/cfg/config.properties
```

Die Konfigurationsdatei ist eine simple Java-property Datei.

Hinweis: Eine Java-property Datei muss im ISO-8859-1 (auch bekannt als ISO-Latin) Character encoding abgelegt sein. Dies Betrifft vor allem vorkommende Umlaute etc. Stellen Sie sicher, dass der Verwendete Text Editor beim Editieren der Konfigurationsdatei dieses Encoding verwendet.

Diese wurde grundsätzlich hierarchisch aufgebaut, um ein einfache Gruppierung verschiedener Bereiche bewerkstelligen zu können.

Die hierarchischen Ebenen werden durch „.“ voneinander getrennt. So ergibt sich eine Baumartige Struktur von konfigurierbaren Werten.

Kommentare können zeilenweise eingefügt werden. Kommentarzeilen beginnen mit dem Raute Zeichen „#“. Kommentarzeilen werden von der Applikation nicht berücksichtigt und können verwendet werden um die Konfigurationsdatei besser lesbar zu machen oder Anmerkungen anzubringen. Beispiel:

```
# Das ist ein Kommentar
```

Es wird empfohlen jede Konfigurationsdatei einleitend mit einem Kommentar zu versehen, welches Auskunft über die Herkunft der Konfiguration, den Verwendungszweck und die darin definierten Profile gibt. Zum Beispiel:

```
# PDF-AS Konfigurationsdatei für den Gebrauch im Amt XYZ.
```

Konfigurierbare Bereiche

Applikations bzw. Modul Spezifikation

Der zu verwendende Normalisierungsalgorithmus für Textsignaturen muss in jeder Konfigurationsdatei explizit gesetzt werden. Gegenwärtig ist dieser auf den Wert V01 zu setzen.

```
normalizer.version=V01
```

Werte für Rekonstruktion des Signatur Blocks: Bezeichnung des PublicKey Algorithmus: Je nach dem im jeweiligen Zertifikat angegebenen Algorithmus (RSA oder ECDSA) werden die Werte der folgenden Keys an geeigneter Stelle in die Verifikations-Templates eingesetzt. Gegenwärtig sind diese Werte folgendermaßen zu setzen:

```
cert.alg.rsa=http://www.w3.org/2000/09/xmldsig#rsa-sha1
cert.alg.ecdsa=http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1
```

Mittels Strict Mode kann die Überprüfung der Version von Eingangsdokumenten an- (true) bzw. ausgeschaltet (false) werden. Ist Strict Mode aktiviert, so akzeptiert die Applikation nur PDF Dokumente der PDF Version 1.4 oder niedriger und weist höher versionierte PDF Dokumente mit einem Fehler zurück. Es ist von Anwendungsgebiet zu Anwendungsgebiet der Applikation unterschiedlich, ob Strict Mode aktiviert sein soll/muss. Standardmäßig ist Strikt Mode deaktiviert, was bedeutet, dass auch PDF Dokumente der Version 1.5 und höher akzeptiert werden.

```
strict_mode=false
```

Basiseinstellungen für die Signaturdienste

Als Signaturdienste (Connectors) werden jene externen Applikationen bezeichnet, welche das Signieren und Verifizieren von Daten ermöglichen. Beispiele sind die Bürgerkartenumgebung (BKU), das Modul für Online Applikationen (MOA) oder die A1 Signatur (A1).

Die Basiseinstellungen des jeweiligen Dienstes werden für den jeweiligen Dienst unter dem Schlüsselelement abgelegt, welches dem Kürzel des Dienstes entspricht. Z.B.: `bku` für BKU, `moa` für MOA, etc. Unter dieser Wurzel werden Eigenschaften/Werte für den jeweiligen Dienst gesetzt.

1.1.1.1 Verfügbarkeit eines Signaturdienstes

Nicht alle Signaturdienste können sowohl in der Web Applikation als auch in der Kommandozeilenapplikation verwendet werden. Mittels der Parameter `available_for_web` bzw. `available_for_commandline` wird angegeben, ob der jeweilige Dienst für die Web Applikation bzw. die Kommandozeile geeignet ist. Durch Setzen des Wertes auf `true` wird der Dienst als geeignet gekennzeichnet. Jeder andere Wert (z.B.: `false`) wird als ungeeignet interpretiert und der Dienst steht dann dort nicht zur Verfügung.

Hinweis: Die Verfügbarkeit eines Signaturdienstes ist quasi eine statische Eigenschaft des jeweiligen Dienstes und kann nicht mittels erneuter Angabe in einem Profil überschrieben werden. Siehe dazu Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** „**Fehler! Verweisquelle konnte nicht gefunden werden.**“.

```
bku.available_for_web=true
bku.available_for_commandline=true
a1.available_for_web=false
a1.available_for_commandline=false
moa.available_for_web=true
moa.available_for_commandline=true
```

1.1.1.2 Basiseinstellungen für die Anbindung an die Bürgerkartenumgebung (BKU)

Signierprozess: Request-Url, Pfad des Signier Request-Template

```
bku.sign.url=http://127.0.0.1:3495/http-security-layer-request
bku.sign.request=./templates/BKUSignRequestB64.xml
```

Der beim Signieren zu verwendende BKU KeyboxIdentifier muss folgendermaßen angegeben werden:

```
bku.sign.KeyboxIdentifier=SecureSignatureKeypair
```

Prüfprozess: Request-Url, Pfad des Prüf Request-Template, Verify Template, SignedProperties Template

```
bku.verify.url=http://127.0.0.1:3495/http-security-layer-request
bku.verify.request=./templates/BKUVerifyRequest.xml
bku.verify.template=./templates/BKUVerifyTemplateB64.xml
bku.verify.template.SP=./templates/BKUVerifyTemplateSP.xml
```

1.1.1.3Basiseinstellungen für die Anbindung an MOA-SS und MOA SP

Signierprozess: Request-Url, Pfad des Signier Request-Template

```
moa.sign.url=http://129.27.153.100:18080/moa-spss/services/SignatureCreation
moa.sign.request=./templates/MOASignRequestB64.xml
```

Der zu verwendende Sign Key von MOA muss folgendermaßen angegeben werden:

```
moa.sign.KeyIdentifier=TestKey2
```

Prüfprozess: Request-Url, Pfad des Prüf Request-Template, Verify Template, SignedProperties Template

```
moa.verify.url=http://129.27.153.100:18080/moa-spss/services/SignatureVerification
moa.verify.request=./templates/MOAVerifyRequest.xml
moa.verify.template=./templates/MOAVerifyTemplateB64.xml
moa.verify.template.SP=./templates/MOAVerifyTemplateSP.xml
```

Die von MOA zu verwendende TrustProfileID muss folgendermaßen angegeben werden:

```
moa.verify.TrustProfileID=Test-Signaturdienste
```

1.1.1.4Basiseinstellungen für die Anbindung an A1

Signierprozess: Request-Url, Pfad des Signier Request-Template und der KeyboxIdentifier

```
a1.sign.url=https://signatur.a1.net/http-security-layer-request
a1.sign.request=./templates/BKUSignRequestB64.xml
```

Der beim Signieren zu verwendende A1 KeyboxIdentifier muss folgendermaßen angegeben werden:

```
a1.sign.KeyboxIdentifier=SecureSignatureKeypair
```

Prüfprozeß: Request-Url, Pfad des Prüf Request-Template, Verify Template, SignedProperties Template

```
a1.verify.url=https://signatur.a1.net/http-security-layer-request
a1.verify.request=./templates/BKUVerifyRequest.xml
a1.verify.template=./templates/BKUVerifyTemplateB64.xml
a1.verify.template.SP=./templates/BKUVerifyTemplateSP.xml
```

Standardeinstellungen für variable Felder der Binärsignatur

Bei der binären Signatur wird das gesamte Dokument inklusive Signaturblock signiert. Da zum Signierzeitpunkt allerdings noch nicht alle Werte verfügbar, also variabel sind werden für binäre Signaturen im Signaturblock (und im signierten Dokument) Platzhalter freigelassen (beispielsweise für Signaturwert, Aussteller, SIG_ID, etc.).

Textsignaturen benötigen diesen Mechanismus nicht. Die phlength Werte werden bei Textsignaturen daher nicht verwendet.

Die Standardlängen (in Bytes) für variable Felder müssen für alle verpflichtenden Felder (Required-Fields) angegeben werden. Ein Wert kleiner gleich 0 gibt an, dass das entsprechende Feld nicht variabel ist, sondern ein statischer Inhalt angezeigt werden soll. Sollte der statische Inhalt des entsprechenden Feldes (`.value`) leer oder nicht definiert sein, so wird das Feld als ganzes nicht angezeigt.

Es ist nicht sinnvoll für zur Prüfung verwendete variable Felder statische Werte anzugeben! Die für eine Prüfung erforderlichen Werte lauten: SIG_DATE, SIG_NUMBER, SIG_ISSUER, SIG_VALUE, sowie SIG_ID. SIG_ID wird aber nur dann verwendet, und scheint auch nur dann im Signaturblock auf, wenn mittels BKU signiert wurde.

Einen Sonderfall stellt das `SIG_KZ` Feld dar. Für alte PDF-AS Textsignaturen, die nicht der Spezifikation unterliegen wird dieses Feld beim Extrahieren der alten Signaturblöcke aus dem Text ignoriert, weil alte Signaturen noch keine URN Kennzeichnung enthalten. Für neue Signaturen (textuell und binär) muss dieses Feld definiert sein. Der Wert dieses Feldes wird zum Signierzeitpunkt vom verwendeten Algorithmus eingesetzt und repräsentiert die URN des Algorithmus laut Spezifikation.

```
defaults.phlength.SIG_DATE=50
defaults.phlength.SIG_NUMBER=50
defaults.phlength.SIG_ISSUER=150
defaults.phlength.SIG_VALUE=200
defaults.phlength.SIG_ID=90

defaults.phlength.SIG_NAME=150
```

Hinweis: Für statische oder selbst definierte Felder sollten aus Gründen des besseren Verständnisses keine Feldlängen definiert werden.

Einstellungen für die Responsemeldungen von BKU und MOA

Die Einstellungen unter dem signature.response Baum geben den Text an, welcher für Prüfergebnisse angezeigt wird, wenn die Prüfapplikation keinen diesbezüglichen Text mitliefert. Es wird empfohlen diese Passagen unverändert zu lassen.

```
signature.response
```

Amtseigenschaften OIDs

Zertifikate können optionale Felder enthalten, welche zusätzliche Information über den Zertifikatsinhaber bereitstellen. Solche Felder finden sich beispielsweise für die amtlichen eigenschaften „Verwaltungseigenschaft“ und „Dienstleistereigenschaft“. Laut X.509 werden zusätzliche Felder über ihre Object Identifiers (OIDs) identifiziert. Im Prüfergebnis werden jene Felder angezeigt, deren OID mit `oid.root` beginnt. Der jeweilige im Prüfergebnis angezeigte Text kann mittels `oid.<OID mit „_“ statt „.“>` angegeben werden.

```
oid.root=1.2.40.0.10
oid.1_2_40_0_10_1_1_1=Verwaltungseigenschaft
oid.1_2_40_0_10_1_1_2=Dienstleistereigenschaft
```

Definition der Signatur Blöcke (Signatur Typen)

Die default Einstellung wird im command line Tool sowie in der Web Applikation vorausgewählt.

Mit `on` oder `off` kann der jeweilige Typ aktiv oder inaktiv gesetzt werden. Diese Definitionsliste wird auch im Webinterface angeboten, wobei nur aktive Definitionen berücksichtigt werden. Der Default-Wert wird dabei vorselektiert. Zum Beispiel:

```
sig_obj.type.default=bmi_su_zmr
sig_obj.types.egov_graz_gv_at=on
sig_obj.types.a_sit_demo=on
sig_obj.types.bmi_su_zmr=on
```

Spezifikation der Signatur Blöcke

Die Spezifikation legt fest, welche Bezeichner (key) im Signatur Block vorhanden sind und welche Werte (value) vordefiniert werden können. Bei der Erzeugung eines Signatur Blocks werden die notwendigen Daten (values) aus der Signaturanfrage Antwort von BKU oder MOA extrahiert und entsprechend zugeordnet. Die Spezifikation sagt jedoch nicht aus, in welcher Reihenfolge oder in welcher Anordnung die Bezeichner und Werte im Signatur Block eingetragen werden. Dafür gibt es eine separate Definition.

Alle Bezeichner die für eine Rekonstruktion des Signatur Blocks notwendig sind, müssen im Signatur Block vorkommen, um als solcher erkannt zu werden.

Der Wert der Description wird im Hilfetext der commandline sowie im WebUI in der Auswahlliste angezeigt (wenn der Typ aktiv ist):

```
sig_obj.egov_graz_gv_at.description=EGOV Graz.gv.at
```

<u>Definition der Bezeichner im Einzelnen:</u>

Name des Zertifikat Inhabers:

```
sig_obj.egov_graz_gv_at.key.SIG_NAME=Inhaber
```

Signaturdatum:

```
sig_obj.egov_graz_gv_at.key.SIG_DATE=Datum
```

Name des Ausstellers:

```
sig_obj.egov_graz_gv_at.key.SIG_ISSUER=Aussteller
```

Angabe des Signatur Verfahrens:

```
sig_obj.egov_graz_gv_at.key.SIG_TYPE=Verfahren
```

Der Signatur Wert:

```
sig_obj.egov_graz_gv_at.key.SIG_VALUE=Signaturwert
```

Id-s der BKU zur Rekonstruktion des Signatur Blocks:

```
sig_obj.egov_graz_gv_at.key.SIG_ID=Kennzeichnung
```

Seriennummer des Zertifikates:

```
sig_obj.egov_graz_gv_at.key.SIG_NUMBER=Seriennummer
```

Metadaten zum Zertifikat:

```
sig_obj.egov_graz_gv_at.key.SIG_META=Hinweis:
```

Kennzeichnung:

```
sig_obj.egov_graz_gv_at.key.SIG_KZ=Algorithmus
```

Definition der default Werte:

```
sig_obj.egov_graz_gv_at.value.SIG_TYPE=urn:publicid:egov.graz.gv.at:AS+bescheid+tb-1.0
sig_obj.egov_graz_gv_at.value.SIG_LABEL=file:///C:/pdf-as/images/amtssignatur_graz.gif
sig_obj.egov_graz_gv_at.value.SIG_META=Hinweis: Dieses Dokument ist nur in elektronischer
Form gültig!
```

Die so genannten Required-Fields SIG_VALUE, SIG_DATE, SIG_ISSUER, SIG_NUMBER SIG_ID sowie SIG_KZ (Signaturwert, Signaturdatum, Aussteller, Seriennummer, BKU IDs falls benötigt sowie Algorithmus Kennzeichnung) müssen in jedem Profil definiert sein. Die Werte dieser Felder sind variabel. Selbst definierte Felder werden hingegen als statisch bezeichent.

Das SIG_NAME (Name des Signators) Feld ist ebenfalls variabel, muss aber nicht in jedem Profil vorhanden sein. Es kann verwendet werden um den Namen des Signators aus dem Signaturzertifikat in den Signaturblock einzufügen.

Default Werte für variable Felder werden ignoriert, da die Werte dieser Bezeichner immer aus den aktuellen Signaturdaten entnommen werden. Zu beachten ist, dass für jedes verwendete variable Feld auch eine entsprechende Feldlänge für die Binärsignatur definiert sein muss (siehe Kapitel „Standardeinstellungen für variable Felder der Binärsignatur“ sowie Kapitel „Profileigene Werte für Anbindung bzw. Feldlängen“).

Hinweis: Es ist absolut notwendig für ein statisches Feld ein entsprechendes value (default Wert) anzugeben, weil der Layoutalgorithmus statische Felder ohne Value verwirft.

Design des Signatur Blocks (Tabelle)

Eine Signatur Block Tabelle besteht aus mindestens einer `main` Tabelle. Die Tabellen Reihen werden steigend durchnummeriert. Der Wert einer Zeile gibt an, was in dieser Zeile dargestellt werden soll.

D.h. Es werden die Synonyme der Bezeichner (laut Signatur Typen Spezifikation) eingetragen. Eine Ausnahme bildet das Synonym `TABLE`. Dieses verweist auf eine eingebettete Tabellendefinition (z.B. info).

Mit Hilfe der Felder –i(Image), -c(Caption=key), -v(Value=value) werden die jeweiligen Werte in die Tabellenzelle eingefügt. Die Trennung von Tabellenzellen erfolgt mit dem Zeichen „|“

```
# Signatur Tabellen Spezifikation
```

Z.B. Definition einer zweispaltigen Tabelle: links das Bild, rechts die Subtabelle info:

```
sig_obj.egov_graz_gv_at.table.main.1=SIG_LABEL-i|TABLE-info
sig_obj.egov_graz_gv_at.table.main.2=SIG_VALUE-cv
sig_obj.egov_graz_gv_at.table.main.3=SIG_META-v
sig_obj.egov_graz_gv_at.table.main.4=SIG_ID-cv
```

Verhältnis der Aufteilung der Tabellen-Spalten:

```
sig_obj.egov_graz_gv_at.table.main.ColsWidth=1 4
```

Styledefinitionen→diese Vererben sich auch auf die Zellen:

Hintergrundfarbe:

```
sig_obj.egov_graz_gv_at.table.main.Style.bgcolor=222 222 200
```

Hinweis: Wenn ein Bild nicht transparent ist, so sollte die Hintergrundfarbe gleich der Bildhintergrundfarbe sein, um unschöne Farbsprünge zu vermeiden.

Innenabstand:

```
sig_obj.egov_graz_gv_at.table.main.Style.padding=3
```

Horizontalausrichtung:

```
sig_obj.egov_graz_gv_at.table.main.Style.halign=center
```

Vertikalausrichtung:

```
sig_obj.egov_graz_gv_at.table.main.Style.valign=middle
```

Rahmenstärke:

```
sig_obj.egov_graz_gv_at.table.main.Style.border=0.1
```

Schriftart: `face, height, weight`

```
default_font: HELVETICA,8,NORMAL
font_face: HELVETICA | TIMES_ROMAN | COURIER
font_height: float value
font_weight: NORMAL | BOLD | ITALIC | BOLDITALIC | UNDERLINE | STRIKETHRU
sig_obj.egov_graz_gv_at.table.main.Style.font=HELVETICA,12,NORMAL
```

Definition einer Subtabelle – zum Beispiel:

```
sig_obj.egov_graz_gv_at.table.info.1=SIG_TYPE-cv
sig_obj.egov_graz_gv_at.table.info.2=SIG_DATE-cv
sig_obj.egov_graz_gv_at.table.info.3=SIG_NAME-cv
sig_obj.egov_graz_gv_at.table.info.4=SIG_ISSUER-cv
sig_obj.egov_graz_gv_at.table.info.5=SIG_NUMBER-cv
sig_obj.egov_graz_gv_at.table.info.ColsWidth=1.5 4
```

Profileigene Werte für Anbindung bzw. Feldlängen

Die Standardwerte für die Anbindungen an MOA, BKU und A1 können für jedes Profil separat überschrieben werden:

```
sig_obj.egov_graz_gv_at.moa.sign.KeyIdentifier=BjornKey
sig_obj.a_sit_demo.moa.sign.KeyIdentifier=TRTestKey
sig_obj.a_sit_demo.moa.verify.url=http://129.27.153.100:18080/moa-
spss/services/SignatureVerification
```

Gleiches gilt für die Längen der variablen Werte:

```
sig_obj.egov_graz_gv_at.phlength.SIG_NUMBER=250
sig_obj.egov_graz_gv_at.phlength.SIG_NAME=250
sig_obj.egov_graz_gv_at.phlength.SIG_ISSUER=300
```

Hinweis: Für ein Profil, welches dazu gedacht ist den Namen des Signators (SIG_NAME) aus dem Zertifikat auszulesen und in den Signaturblock zu integrieren, muss natürlich auch dem SIG_NAME Feld eine Länge zugewiesen werden. Entweder geschieht dies bereits in den Default Einstellungen (was nicht empfehlenswert ist, weil dann diese Länge für alle Profile, also auch für solche, die gar keinen variablen Namen verwenden möchten, gelten würde) oder besser mittels profileigenen Wert.

Positionierung von Signaturblöcken

Ein Signaturblock kann entweder automatisch (Standardeinstellung) oder manuell positioniert werden.

Bei der automatischen Positionierung wird der Signaturblock auf die erste freie Stelle nach dem gesamten Dokumenttext einschließlich der Fußzeile platziert. Sollte auf der letzten Seite nicht mehr genügend Platz dafür sein, so wird eine neue Seite angelegt und der Signaturblock auf dieser platziert.

Mittels manueller Positionierung kann in die Positionierung des Signaturblocks eingegriffen werden. Ein Signaturblock kann auf mehrere Arten manuell positioniert werden:

```
-pos x:x_algo;y:y_algo;w:w_algo;p:p_algo;f:f_algo


 x_algo := 'auto' ... automatische Positionierung
        :=        ... Absolutwert für x-Position

        Default bei Fehlen des Paramters: 'auto'

 y_algo := 'auto' ... automatische Positionierung
        := 'autofl' ... auto mit Fußzeile
        :=        ... Absolutwert für y-Position

        Default bei Fehlen des Paramters: 'auto'

 w_algo := 'auto' ... automatische Breite
        :=  +[0..9] ... Absolutwert für Breite

        Default bei Fehlen des Paramters: 'auto'

 p_algo :=  'auto' ... Automatisch letzte Seite
        :=  'new'  ... Neue Seite am Ende des Dokumentes
        :=  +[0..9] ... Seitennummer

        Default bei Fehlen des Paramters: 'auto'

 f_algo :=  +[0..9] ... y-Offset für Footer

        Default bei Fehlen des Paramters: '0'
        Wird nur bei y:auto berücksichtigt, andernfalls ignoriert!
```

Default ist somit: `-pos x:auto;y:auto;w:auto;p:auto;f:0`

Variationen zum Beispiel:

```
-pos x:auto;y:auto;w:auto
-pos x:10.0;y:10.0;w:100.0
-pos x:10.0;y:10.0;w:100.0;p:new;f:10
-pos x:22.0;y:auto;w:450.0;p:2;f:25
-pos x:auto;y:auto;w:auto;p:auto;f:25.0
-pos f:25.0
-pos x:150;y:22;w:400
-pos x:10.0;w:155.0
```

Parameter `p` gibt dabei die Seite an, auf welcher der Signaturblock angebracht werden soll. Eine gültige Seitenzahl als Parameter bedeutet, dass der Signaturblock auf der angegebenen Seite absolut positioniert wird. `p=new` bedeutet, dass eine neue, leere Seite an das Dokument angefügt und auf dieser dann eine absolute Positionierung vorgenommen wird. `p=auto` bedeutet, dass die Signatur eigentlich automatisch positioniert werden soll, bei Berechnung des Endes des Textes allerdings die Fußzeile ggf. ausgenommen wird. Mit diesem Mechanismus ist es möglich einen Signaturblock automatisch zwischen Text und Fußzeile zu platzieren, sofern dort genügend Platz vorhanden ist.

Bei absoluter Positionierung geben die Parameter die `x` und `y` die Koordinaten der linken oberen Ecke des Signaturblocks auf der gewählten Seite an. `x` wird von links nach rechts gemessen. `y` wird von unten nach oben gemessen. Der Koordinaten Ursprung liegt in der linken unteren Ecke der Seite. Der Parameter `w` gibt zudem die Breite des Signaturblocks an. Diese wird von links nach rechts gemessen und muss eine Zahl größer als 0 sein.

Bei automatischer Positionierung unter Berücksichtigung der Fußzeile ist der Parameter `y:autofl` zu setzen. Zusätzlich kann dann mit dem Parameter `f` die y Koordinate der Oberkante der Fußzeile angegeben werden. Ist zwischen dem Ende des Fließtextes und der Oberkante der Fußzeile genügend Platz für den Signaturblock, so wird dieser dort platziert. Ansonsten wird der Signaturblock auf eine neue Seite gesetzt.

Alle Koordinaten werden in PDF User Space Einheiten gemessen. Eine hochformatige A4 Seite ist demnach 595 Einheiten breit und 842 Einheiten hoch.

Für weitere Informationen siehe auch den `-pos` Parameter des command line Tools.

Hinweis: Es ist durchaus möglich den Signaturblock so zu positionieren, dass er nicht sichtbar ist. Weiters kann er durch die Wahl einer sehr kleinen Breite unschön entstellt werden. Es liegt in der Verantwortung des Users eine ansprechende Darstellung und vernünftige Werte für die absolute Positionierung zu wählen.

Anmerkungen zur Text- bzw. Binärsignatur

Es ist erwünscht, dass das Aussehen von Signaturblöcken unabhängig vom verwendeten Signaturalgorithmus ist. Um dieses Ziel zu erreichen ist es notwendig über den Hauptunterschied zwischen Textsignatur und Binärsignatur bescheid zu wissen:

Bei der Textsignatur wird der bestehende Text signiert, exklusive des Signaturblocks. Bei der Binärsignatur wird der Signaturblock allerdings mitsigniert. Dies hat zur Folge, dass die Textsignatur den Signaturblock erst erstellt nachdem der Text bereits signiert wurden und sie

sämtliche Signaturdaten bereits zur Verfügung hat, was das Erstellen des Signaturblocks vereinfacht. Die Binärsignatur muss den Signaturblock allerdings vor dem Signaturvorgang erstellen und entsprechend Platzhalter für variable Felder freilassen, welche erst nachträglich befüllt werden.

Daher ist es auch immer notwendig die phlength Platzhalter für alle variablen Felder im Signaturprofil anzugeben (siehe Kapitel „Profileigene Werte für Anbindung bzw. Feldlängen“ und „Standardeinstellungen für variable Felder der Binärsignatur“), damit die Binärsignatur weis wie viel Platz sie freilassen muss.

Ein nicht als variabel markiertes Feld wird von der Binärsignatur automatisch als statisch betrachtet und unterliegt somit den Formatierungsgesetzen für statische Felder (default Wert). Schlagend wird dieser Umstand eigentlich nur beim optionalen Feld SIG_NAME, mit welchem der Name des Signators eingefügt werden kann. Alle anderen Required-Fields müssen ohnehin als variabel ausgewiesen werden um die Prüfbarkeit zu gewährleisten.

Für das Feld SIG_NAME wird in der Textsignatur immer der Name des Signators eingesetzt, weil die Signatordaten ja zum Zeitpunkt der Erstellung des Signaturblocks bereits vorhanden sind. Ein korrektes Profil enthält deshalb auch eine phlength Angabe (am besten profileigen) für SIG_NAME, damit die Binärsignatur einen entsprechenden Platzhalter für SIG_NAME freilässt und sich damit gleich verhält wie die Textsignatur. Wird im Binärsignaturfall auf phlength vergessen, so gilt SIG_NAME als statisches Feld und wird daher nach den Regeln für statische Felder behandelt.

Konfiguration help_text.properties

Die Konfigurationsdateien befinden sich in:

```
PDF-AS/cfg/help_text.properties
PDF-AS/cfg/help_text.properties
```

Die Konfigurationsdatei ist eine simple java-property Datei (siehe „Konfiguration config.properties“).

Diese wurde grundsätzlich hierarchisch aufgebaut, um ein einfache Gruppierung verschiedener Bereiche bewerkstelligen zu können.

Die hierarchischen Ebenen werden durch „.“ voneinander getrennt. So ergibt sich eine Baumartige Struktur von konfigurierbaren Werten.

Die Strukturebenen unterscheiden sich in der letzten Dezimalzahl. Zahlen mit derselben Hunderterstelle, sollen als zusammengehörig aufgefasst werden.

Es wird dabei unterschieden, welche Exception an den Benutzer weitergereicht werden soll. D.h. in dieser Konfiguration sind lediglich Texte deklariert, die teilweise für eine Klasse von Problemen gelten. Wenn externe Fehler (Fehler von externen Applikationen) auftreten, werden diese direkt an das UserInterface weitergereicht.

Beispiel:

```
error.code.10=Das System kann nicht initialisiert werden.
```

Kommentare können zeilenweise eingefügt werden. Kommentarzeilen beginnen mit dem Raute Zeichen „#“. Beispiel:

```
# Das ist ein Kommentar
```

Zertifikate

Zertifikate werden einerseits beim Erstellen des Signatur Blocks eingesetzt und andererseits zum Verifizieren des Signatur Blocks als Referenz herangezogen.

Zertifikate sind in einem eigenen Ordner abgelegt:

```
PDF-AS/certificates
```

In diesem Verzeichnis sind alle verfügbaren Zertifikate in speziellen Unterordnern abgelegt. Diese Unterordner werden automatisch erzeugt, wenn Zertifikate beim Starten der Applikation in das System geladen werden. Dazu müssen Zertifikate im Ordner „tobeadded“ abgelegt werden.

Nur gültige X509Certificate Dateien im base64-kodiertne `.cer` Format werden erkannt (so wie es auch die Windows-Zertifikatsverwaltung zum Abspeichern anbietet), aufbereitet und im zugehörigen Zertifikats-Store gespeichert. Alle anderen Formate die interpretiert werden können, werden zwar übernommen, führen jedoch bei der Überprüfung zu falschen Ergebnissen!

Zertifikats-Store

Ein Zertifikats-Store ist ein Ordner mit einer speziellen Namensgebung:

Der Aussteller eines Zertifikates (IssuerName) wird normalisiert, alle white spaces entfernt, ein MD5-Hash gebildet und diesen in einen Base64 codierten String gewandelt. Das Zeichen „/“ wird durch „_“ ersetzt. D.h. sind mehrere Zertifikate eines Ausstellers vorhanden, so sind diese in einem Ordner zusammengefasst. Der Name der Zertifikatsdatei wird dabei geändert auf „Seriennummer.der“ im Format „Base64“.

Wird ein PDF Dokument mit der Applikation signiert, wird das Zertifikat des SignaturResponse im zugehörigen Zertifikats-Store abgelegt.

Beispiel für gültige Zertifikate:

```
PDF-AS/certificates/bIDzucmqb7EAp5QhXpLLyC0Pl28=/173416956699810981094293464745 98.der
PDF-AS/certificates/38uXxn1yOSVtAjoxv+mBUZMd7wQ=/65090.der
PDF-AS/certificates/8nGPZXpcp2rrP0ogDdscGCoGigg=/29.der

Z.B: bIDzucmqb7EAp5QhXpLLyC0Pl28= ➔ Base64(“CN=VSigCA2,O=Hauptv.
österr.Sozialvers.,C=AT”)
```

command line Tool

Allgemeines

Um das command line Tool starten zu können, müssen alle notwendigen Libraries im classpath mit aufgenommen werden. Es gibt eine Batch Datei „pdf-as.bat“ die alle Aufrufe kapselt.

Mit dem command line Tool können vom Prinzip her lediglich die zwei Use-Cases, Dokument signieren und Dokument prüfen, abgearbeitet werden.

Für jedes zu verarbeitende Dokument muss die Applikation erneut gestartet werden. Die Übergabe eines Ordners zur Verarbeitung ist in dieser Version nicht vorgesehen. Da aber aufgrund der Konfiguration beim Signieren verschiedene Signatur Blöcke erstellt werden können und auch mehrere externe Applikationen verwendet werden können, ergeben sich durchaus mehr Anwendungsfälle.

Durch Verwendung einer Log4j Konfigurationsdatei (`PDF-AS/cfg/log4j.properties`) kann die Verbosität der Textausgabe gesteuert werden.

Mit dem Parameter `-mode` (`sign` oder `verify`) kann der entsprechende Workflow aktiviert werden. Dies entspricht der Umsetzung der zwei Use Cases.

Mit dem Parameter `-connector` wird die Applikation zum Signieren oder Prüfen (z.B.: `bku` oder `moa`) ausgewählt.

Wird gar keine oder eine falsche Eingabe getätigt, so erscheint die Benutzererklärung (usage) des command line Tools.

Signieren

Mit dem Parameter `-sigmode` wird der Algorithmus ausgewählt, welcher die Signatur durchführen soll (`-binary` oder `-textual`).

Der Parameter `-sigtype` erlaubt es, das Profil auszuwählen.

Die Parameter `-username` und `-password` geben die bei der Signierung zu verwendende Benutzer Identifizierung an. Diese Parameter sind optional.

Mittels `-pos` kann die Position des Signaturblocks absolut angegeben werden. Für die Positionsangabe per Kommandozeile gelten die selben Regeln wie für die Positionsangabe in einem Signaturprofil (siehe Kapitel „Positionierung von Signaturblöcken“), Die mittels `-pos` Parameter angegebene Position überschreibt die im ausgewählten Profil angegebene Positionen.

Verifizieren

Der Parameter `-verify_which` ermöglicht es explizit einen Signaturblock auszuwählen, der signiert werden soll. Der Wert muss nullbasierend, ganzzahlig sein. Wird dieser Parameter nicht angegeben, so werden alle Signaturen geprüft.

Eingabe- und Ausgabedokument

Nach den Parametern wird das Eingabedokument spezifiziert.

Optional kann auch ein Ausgabedokument angegeben werden, in welches im Falle des Signierens das signierte Dokument geschrieben wird.

Prozesse / Workflow

Der Workflow gliedert sich in folgende Schritte:

UC1 PDF-Dokument signieren:

1. Dokument Checks, Format etc.
2. Signaturalgorithmus aufrufen (Vorbereitung)
3. Die zu signierenden Daten mittels externer Applikation signieren
4. Signaturalgorithmus aufrufen (Nachbereitung)
5. Signiertes Dokument schreiben.

UC2 PDF-Dokument überprüfen:

1. Dokument Checks, Format, etc.
2. Signaturblöcke extrahieren (binäre wie textuelle)
3. Alle, oder nur den ausgewählten Signaturblock mittels externer Applikation prüfen
4. Prüfresultat auf Kommandozeile ausgeben.

Aufrufparameter und Konfiguration

Die Konfigurationsdateien befinden sich im Pfad „cfg“:

```
PDF-AS/cfg/config.properties
PDF-AS/cfg/help_text.properties
PDF-AS/cfg/log4j.properties
```

Die Logdatei von Log4j wird in folgender Datei abgelegt (Änderung möglich in `log4j.properties`):

```
PDF-AS/logs/pdf-as.log
```

Um alle Aufrufparameter des command line Tools ausgeben zu können, kann einfach die Batch Datei gestartet werden:

```
pdf-as.bat
Please specify a mode ('-mode' parameter).
Usage: pdf-as [OPTIONS] <input file> [output file]
  Required OPTIONS:
    -mode <sign|verify>
      sign ... signs a document
      verify ... verifies a document
    -connector moa|bku|
      moa ... MOA
      bku ... BKU
  OPTIONS for signation:
    -sigmode <binary|textual>
      binary ... signs the complete binary document
      textual ... signs only the textual portion of the document
      detached ... signs the document and returns the xml signature of it.
    -sigtype <CIO-BUND2|ECDL|TEST-BBB|test_wlackner|egov_graz_gv_at|CL-DEMO-
PROFIL|bmi_su_zmr|a_sit_demo2|TEST-AAA|a_sit_demo|CIO-BUND-neu|profEGIZ|CIO-
BUND3|ASIT-DEFAULT|DEMO-PROFIL>
      ... [optional] the profile to be used. If omitted, the default
          profile is used.
      CIO-BUND2 ... CIO des BUNDES 2
      egov_graz_gv_at ... (default) EGOV Graz.gv.at
      bmi_su_zmr ... Bundesministerium für Inneres ZMR
      a_sit_demo2 ... A-Sit Demo 2 (semantisch gleich A-Sit Demo 1)
    -username <user_name> ... [optional] the user name
    -password <password> ... [optional] the user password
    -pos <position> ... [optional] the position of the signature block
  OPTIONS for verification:
    -verify_which <number> ... [optional] zero based number of the signature
      to be verified. If omitted, all signatures are verified.
  Example usage:
    pdf-as -mode sign -connector moa some_document.pdf
    pdf-as -mode verify some_document.pdf_out.pdf
```

Start des command line Tools Beispiele:

```
pdf-as.bat –mode sign –connector moa –sigmode binary –username xxx –password yyy document4.pdf

pdf-as.bat –mode sign –connector bku –sigmode textual -username=xxx –password yyy
document4.pdf_out.pdf

pdf-as.bat –mode verify –connector moa -verify_which 0 document4.pdf_out.pdf

pdf-as.bat –mode verify –connector moa document4.pdf_out.pdf_out.pdf
```

Web Anwendung

Allgemeines

Mit Hilfe der Web Anwendung können mittels eines Browsers die beiden Use Cases signieren und verifizieren von PDF-Dokumenten abgearbeitet werden. Die dafür notwenige Infrastruktur muss entsprechend vorhanden sein. Dies ist eine Servlet-Engine z.B. Tomcat um die eingehenden Browser Requests verarbeiten zu können und die notwendige Dokumentstruktur im Webserver. Bei der Auslieferung existiert bereits ein vorkonfigurierter Tomcat Server, indem alle notwendigen Daten vorhanden sind.

Der Einstieg in die Web Anwendung ist mit http://localhost:8080/pdf-as/ entsprechend vorkonfiguriert.

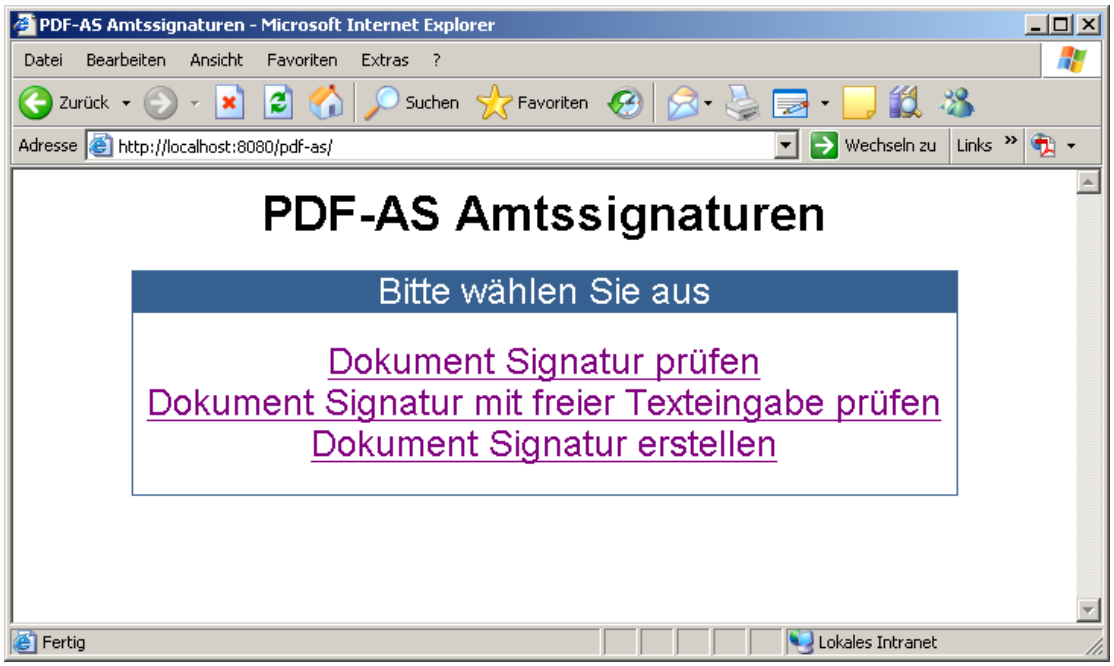

Abbildung 1: Einstiegsdialog

Bereits im Einstiegsdialog sind die realisierten Use Cases umgesetzt.

Prüf-Request

Beim Prüf-Request kann eine PDF-Datei upgeloadet werden und die Applikation eingestellt werden mit der geprüft werden soll. Als Standard ist hier MOA selektiert.

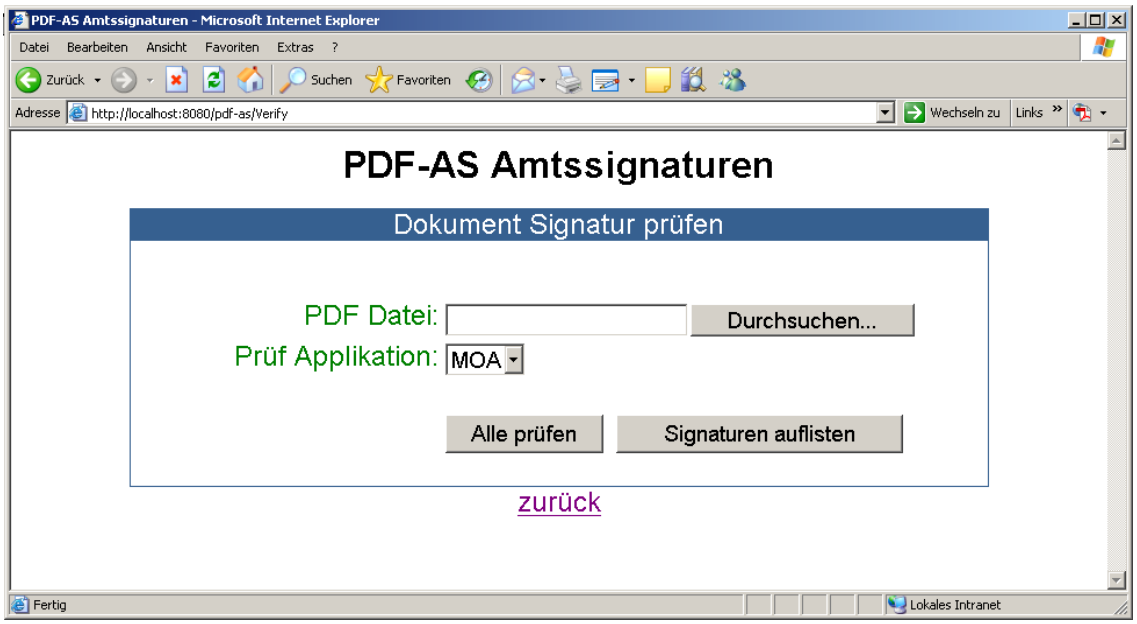

Abbildung 2: Dialog Prüf-Request

Bei Aktivierung der Signaturliste werden alle im Dokument gefundenen Signaturen aufgelistet und können einzeln geprüft bzw. bearbeitet werden.

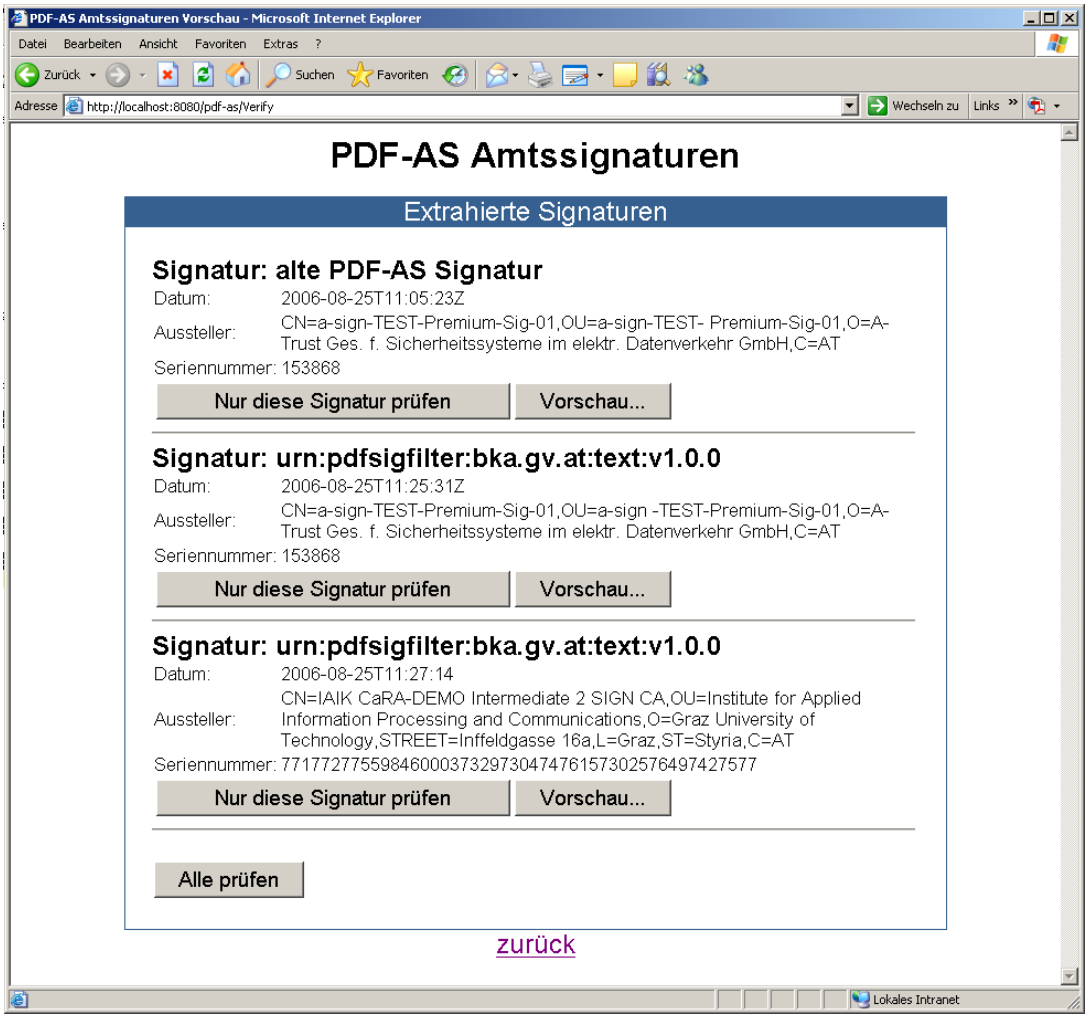

Abbildung 3Liste mit gefundenen Signaturen

Die Vorschau zeigt für binäre Signaturen die signierten Daten. Die textuelle Vorschau zeigt den Dokument Roh Text und den Signatur Block. Bei Aktivierung des Buttons Signatur prüfen, wird der Prüf-Workflow abgearbeitet. Mittels Alle prüfen werden alle Signaturen geprüft.

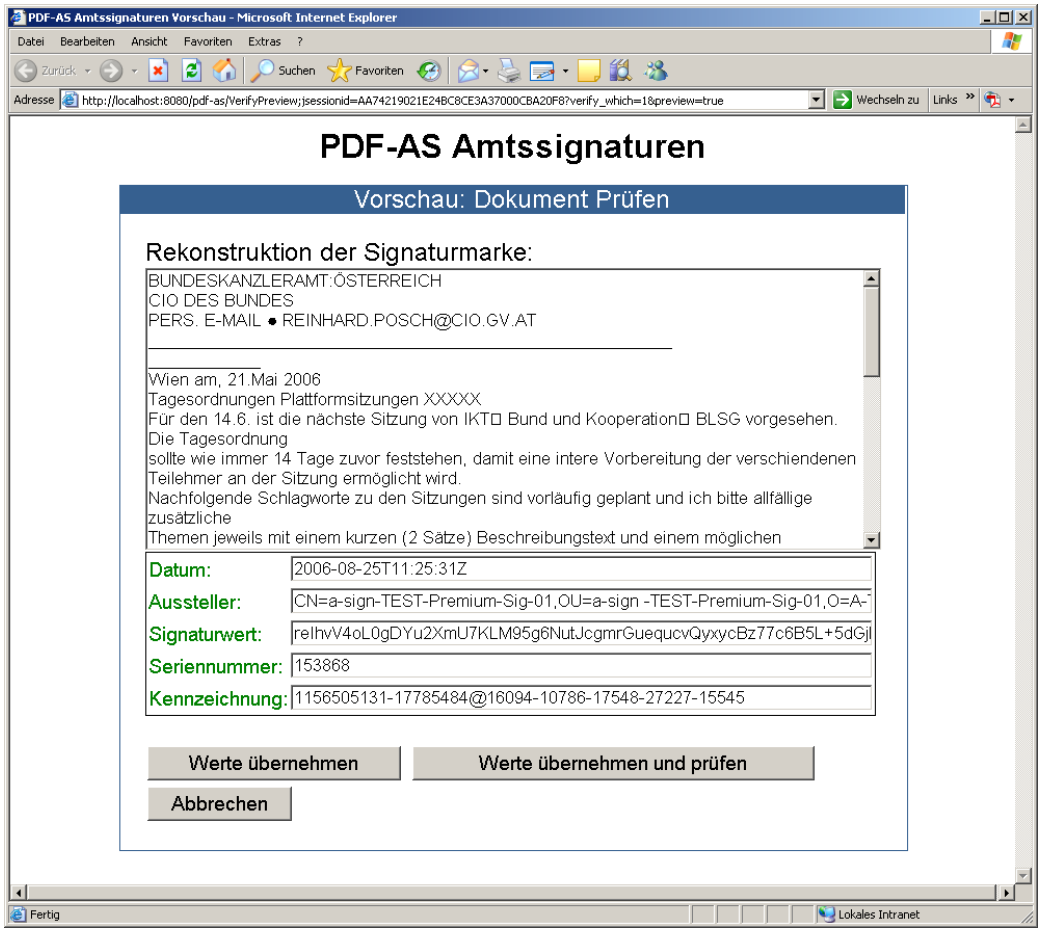

Abbildung 4: Beispiel einer Voransicht des Prüf-Requests

Ein mögliches Ergebnis einer Prüfanfrage ist in Abbilung 5 dargestellt. Es werden dabei drei Bereiche unterschieden. Im Zertifikatsbereich werden die Daten des Zertifikats aus dem Signatur Block dargestellt und das Ergebnis der Zertifikatsüberprüfung. Bei der Darstellung der jeweiligen Check's ist der Ergebnistext aufgrund des Ergebniscodes mit unterschiedlichen Hintergrundfarben belegt:

Grüne Hintergrundfarbe → Ergebnis korrekt

Gelbe Hintergrundfarbe → Teilergebnisse korrekt

Rote Hintergrundfarbe → Ergebnis nicht korrekt

Der zweite Bereich ist für die Überprüfung des Signaturwert selbst vorgesehen. Der dritte Bereich gibt das Resultat der Manifestprüfung aus. Auch in diesen Bereich ist die farbliche Codierung als visuelle Hilfestellung zu interpretieren.

Die farblichen Codierungen sind über das zentrale Stylesheet geregelt. Anpassungen hierzu sind in folgendem File zu tätigen:

`PDF-AS/css/styles.css`

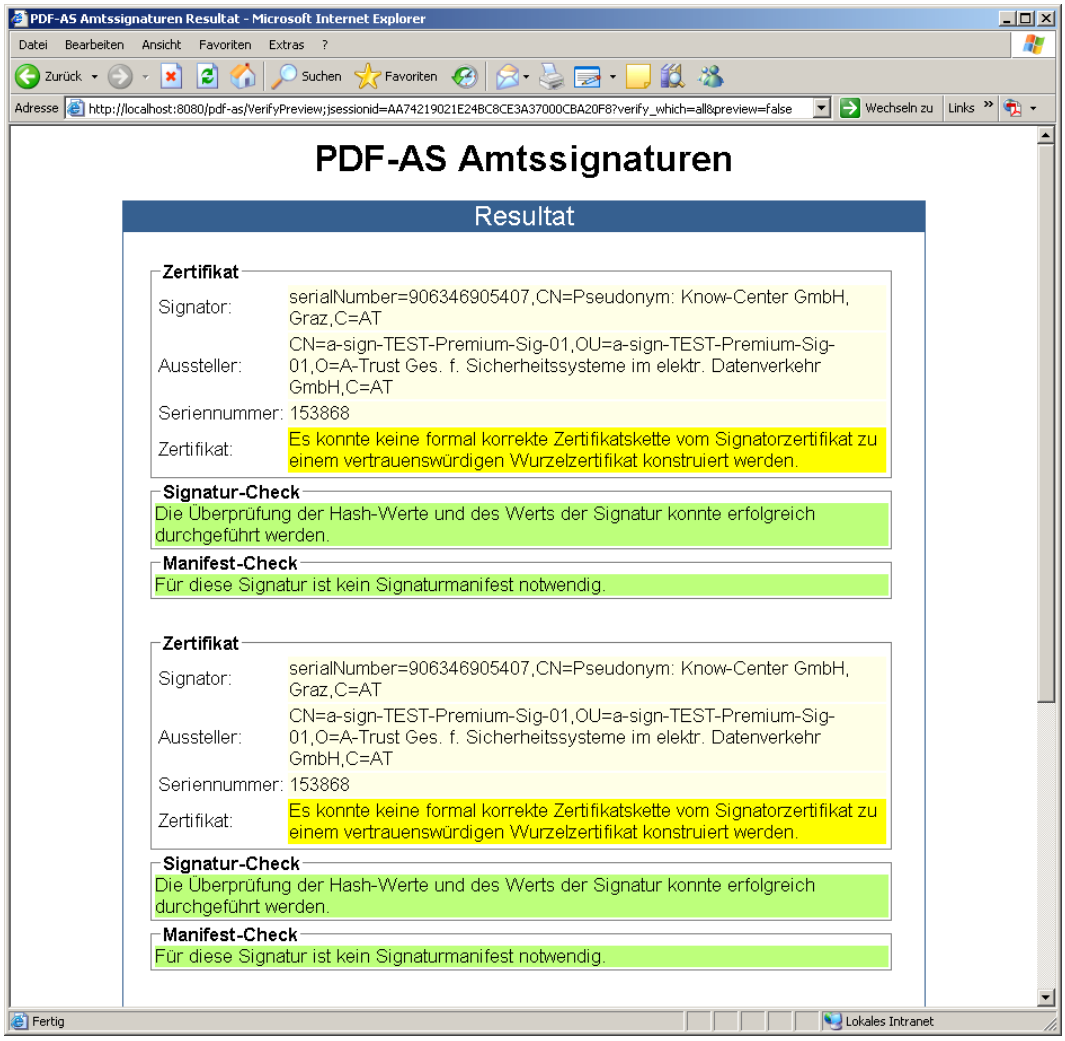

Abbildung 5: Beispiel für das Resultat einer Signaturprüfung

Signier-Request

Um den Dialog zum Signieren eines Dokumentes anzeigen zu können, muss sich ein Benutzer am System mit Benutzername und Passwort anmelden (default: `tomcat/tomcat`).

Beim Signier-Request kann eine PDF-Datei upgeloadet werden, der Typ des Signatur Blocks (Signatur-Typ → der Defaultwert wird hierbei in der Auswahlliste selektiert) ausgewählt werden, die Applikation eingestellt werden mit der signiert werden soll (als Standard ist hier MOA selektiert), und die Darstellungsweise des signiertes Dokumentes (als Download oder direkt im Browser). Der entsprechende Dialog ist in Abbildung 6 ersichtlich. Wiederum ist eine Voransicht des zu signierenden Text zur Überprüfung möglich. In dieser Voransicht kann der Text jedoch nicht mehr verändert werden, auch die Daten eines bereits vorhandenen Signatur Blocks nicht, denn es wird lediglich das bestehende PDF-Dokument um den PDF- Signatur Block erweitert. Der Text wird nicht neu geschrieben!

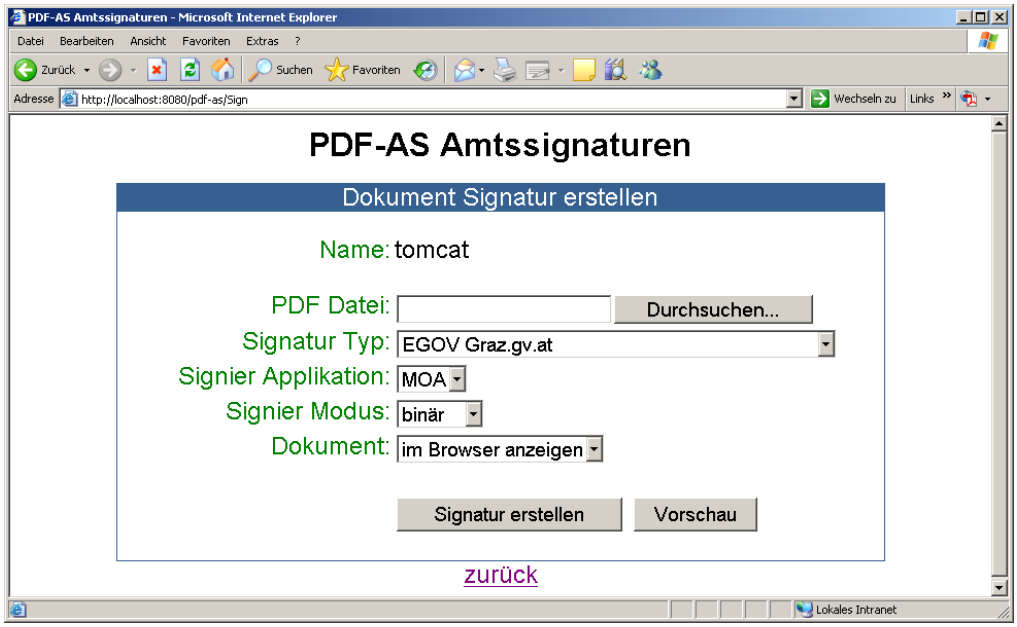

Abbildung 6: Dialog des Signier-Requests

Das erzeugte Dokument wird entweder direkt im Browserfenster angezeigt, oder kann als Download heruntergeladen werden.

Starten und Konfiguration der Web Applikation

Die Konfigurationsdateien befinden sich in:

```
pdf-as-tomcat/webapps/pdf-as/cfg/config.properties
pdf-as-tomcat/webapps/pdf-as/cfg/help_text.properties
pdf-as-tomcat/webapps/pdf-as/WEB-INF/classes/log4j.properties
```

Das Logfile wird standardmäßig in folgendem Verzeichnis abgelegt:

```
pdf-as-tomcat/logs/pdf-as.log
```

Dies ist jedoch wiederum über die Anpassung im File `log4j.properties` veränderbar.

Temporäre Dokumente werden im Ordner `pdf-as-tomcat/webapps/pdf-as/pdfastmp` abgelegt. Dieser Ordner wird beim Starten sowie Stoppen der Applikation komplett geleert.

```
Start/Stop der Web Applikation:
C:\pdf-as-tomcat\bin>catalina start
C:\pdf-as-tomcat\bin>catalina stop
```

http://localhost:8080/pdf-as/

Module

Allgemeines

Alle Module sind in Java (min 1.4.2) erstellt worden. Die Packagestruktur gruppiert die Klassen nicht nach den Modulen selbst, sondern nach deren Einsatzbereich, bzw. logischen Zuordnung.

Eine detaillierte Beschreibung der jeweiligen Klassen ist in der Zugehörigen Source Dokumentation ersichtlich. Diese wurde mit Java-Doc Statements erstellt und ist somit auch in übersichtlicher Weise mittels Webbrowser einsehbar.

Packages

Die wichtigsten Pakete lauten:

```
at.knowcenter.wag.egov.egiz
at.knowcenter.wag.egov.egiz.cfg
at.knowcenter.wag.egov.egiz.commandline
at.knowcenter.wag.egov.egiz.framework
at.knowcenter.wag.egov.egiz.pdf
at.knowcenter.wag.egov.egiz.sig
at.knowcenter.wag.egov.egiz.web
```

Die wichtigsten Klassen werden nachfolgend mit kurzer Beschreibung aufgeführt.

at.knowcenter.wag.egov.egiz

Classes

```
PdfAS – enthält modulübergreifende Hilfsfunktionen
```

at.knowcenter.wag.egov.egiz.cfg

Classes

```
SettingsReader – Hauptklasse zum Auslesen der Konfigurationsdaten
```

at.knowcenter.wag.egov.egiz.commandline

Classes

```
Main – Hauptklasse der Kommandozeilen Applikation
```

at.knowcenter.wag.egov.egiz.framework

Interfaces

```
Signator     - Interface aller Signieralgorithmen
Verificator  - Interface der Signatur-Extraktoren
```

Classes

```
SignatorFactory    – Factory zum Generieren von Signatoren
VerificationFilter – Filter zum Extrahieren von Signaturblöcken
```

```
Subpackages        - Implementationsklassen der jeweiligen Algorithmen
```

at.knowcenter.wag.egov.egiz.pdf

Classes

```
BinarySignature  – Klasse mit Hilfsfunktionen für die Binärsignatur
TextualSignature - Klasse mit Hilfsfunktionen für die Textsignatur
```

at.knowcenter.wag.egov.egiz.sig

Interfaces

```
Connector – Interface der Connectors
```

Classes

```
ConnectorFactory – Factory zum erstellen von Connectoren
Subpackages      - Connector Implementationen
SignatureObject  - Abstrakte Repräsentation einer Signatur
SignatureHolder  - Kapselung von SignatureObject und dem signierten Text.
```

at.knowcenter.wag.egov.egiz.web

Classes

```
Sign   - Signierservlet
Verify - Verifizeirservlet
```

User Interface

Das User Interface Modul hat die Aufgabe die Schnittstelle zwischen der Usereingabe und dem dahinter liegenden Framework zu bilden. Das comman line Tool (Main) ermöglicht es dem User Befehle auf textueller Basis abzugeben. Dies ist vor allem für Stapelverarbeitung interessant. Die Web Applikation stellt eine grafische Oberfläche dar, mit welcher die Applikation bedient werden kann.

Die User Interface Komponenten bedienen sich im wesentlichen folgender drei Hauptmodule: Connector, Signator und Verificator.

Zur Web Applikation ist anzumerken, dass lokale Connectors über den Asynchronous Redirect Mechanismus, asynchron auf Seite des Clients betrieben werden.

Connector

Ein Connector hat die Aufgabe die Anbindung an die jeweilige Externe Applikation MOA oder BKU zu kapseln.

Im Signierfall erhält ein Connector den zu signierenden Text, signiert diesen mittels der gewählten Applikation und gibt ein SignatureObject mit den Singierungsdaten zurück.

Im Prüffall erhält der Connector einen SignatureHolder und gibt das ausgewertete SignatureResponse Objekt zurück.

Ein lokaler Connector (BKU, A1) teilt diese Aufgaben noch weiter in 3 Phasen auf: Vorbereitung, Verbindung, Analyse. In der Vorbereitung wird für die gegebenen Daten eine XML Request formatiert. In der Verbindungsphase wird dieser XML Request and die gegebene URL geschickt. Die so erhaltene XML Antwort wird in der Analysephase wieder vom Connector bearbeitet und in abstrakte Objekte (SignatureObject, SignatureResponse) umgewandelt. Diese Aufteilung ermögliche es, die Verbindungsphase lokal am Client des Users durchzuführen.

Signator

Ein Signator erhält eine PDF Datei und erstellt gemäß seinem Algorithmus ein Ausgabedokument, welches im SignResult gekapselt ist. Ein SignResult enthält die binären Ausgabedaten sowie den zugehörigen MIME Type.

Ein Signator ist in zwei Phasen unterteilt: Vorbereitung und Nachbereitung. In der Vorbereitungsphase wird im Wesentlichen für das gegebene Dokument ein passender SignatureHolder erstellt, welcher dann vom Connector signiert werden kann. In der Nachbereitungsphase werden die Signierungsdaten dann ausgewertet und entsprechende dem Algorithmus in einen SignResult umgesetzt.

BinarySignator v.1.0.0

Erstellt eine Binärsignatur entsprechend der Spezifikation.

Algorithmus:

1. Das gegebene Dokument wird analysiert.
2. Entsprechend dem gewählten Profil wird ein Signaturblock vorformatiert, dessen variable Felder mit Platzhaltern befüllt werden. Durch die Breite des gewählten Platzhalterzeichens („w“) wird das optische Layout des Signaturblocks festgelegt.
3. Durch Analyse des Content Streams der Tabelle sowie sonstiger Daten (/Cert) werden die byte ranges bestimmt.
4. Ein EGIZ Dictionary wird erstellt.
5. Ein Incremental Update Block wird erstellt und an das Dokument angehängt.
6. Die Löcher in den byte ranges werden mit 0 befüllt.
7. Das gesamte Dokument wird base64 codiert und dem Connector übergeben.
8. Nach Erhalt des signierten SignatureObjects werden die variablen Werde unter Berücksichtigung des /encodings in die zuvor freigelassenen Löcher eingesetzt.
9. Das Ausgabedokument ist das gesamte PDF Dokument.

TextualSignator

Erstellt eine Textsignatur entsprechend der Spezifikation.

Algorithmus:

1. Der Dokumenttext wird extrahiert.
2. Der extrahierte Text wird normalisiert und dem Connector übergeben.
3. In der Nachbereitungsphase wird ein Incremental Update erstellt, welches die signierten Daten enthält.
4. Das Ausgabedokument ist das gesamte PDF Dokument.

VerificationFilter

Die Aufgabe des VerificationFilters ist es, für ein gegebenes PDF Dokument alle darin enthaltenen Signaturblöcke in SignatureHolders umzuwandeln, welche dann von einem Connector geprüft werden können.

Algorithmus:

1. Das Dokument wird in seine Incremental Update Blöcke zerlegt.
2. Ein Binärer Block wird mittels BinaryVerificator analysiert.
3. Ein Textueller Block (dies sind alle nicht binären Blöcke) wird mittels TextualVerificator analysiert.
4. Eine Sonderstellung hat der oberste Incremental Update Block: dieser wird, nachdem er mittels TextualVerificator geprüft wurde nochmals auf alte Signaturen durchsucht.

BinaryVerificator

Algorithmus:

1. Das EGIZ Dictionary wird geparst.
2. Mittels /replaces und /encodings Array werden die variablen Signaturwerte wiederhergestellt.
3. Die Löcher in den byte ranges werden wieder mit 0 befüllt. Damit ergibt sich das signierte Dokument.
4. Dieses wird base64 codiert.
5. Daraus ergibt sich der SignatureHolder, welcher dem Connector übergeben wird.

TextualVerificator

1. Der gegebene Text des gesamten Dokuments bis einschließlich des Incremental Updates wird extrahiert und normalisiert.
2. Die Signaturblöcke werden mittels Algorithmus „Absolute Textsignatur“ extrahiert.
3. Im signierten Text des ersten (ältesten) Signaturblocks wird noch nach einer alten PDF-AS Signatur gesucht mittels Algorithmus „End Textsignatur“.

Algorithmus „Absolute Textsignatur“

Algorithmus: findPotentialSignaturesForProfile

Algorithmus zur eindeutigen Erkennung von Textsignaturblöcken für einen gegebenen Text und ein gegebenes Profil:

Ziel: Alle zu diesem Profil gehörigen Blöcke finden (mit Position etc.).

1. Reichenfolge der Captions laut Profil ermitteln.
2. Definition: End Caption = die unterste Caption dieses Profils
3. Vorkommen der End Caption im Text suchen ➔ Liste von Indices der End Captions.
4. Für jede gefundene End Caption:
    a. Den dazu möglichen vollständigen Signaturblock finden (Suche der anderen Captions laut Reihenfolge wie gehabt).
    b. Rückwärtsprüfung durchführen:
        i. Ausgehend von der obersten Caption rückwärts den Text bis zur Endcaption suchen.
        ii. Überprüfen, ob für alle Captions die Positionen (Indizes) gleich sind wie bei der ersten Vorwärtsuche.
            1. Wenn nicht alle Positionen ident sind, so bleibt dieser Fund unberücksichtigt.
            2. Wenn alle Positionen ident sind, so kommt dieser Fund in die Liste der Potentiellen Kandidaten.
5. ➔ Liste der gesichert gefundenen, potentiellen Kandidaten.

Algorithmus: „Absolute Textsignatur“

1. Ermittle die Liste der Profile aus dem Config File.
2. Für jedes Profil:
    a. Finde für den gegebenen Text die Liste der potentiellen Kandidaten mittels findPotentialSignaturesForProfile
3. ➔ Liste aller gefundenen Kandidaten
4. Sortiere diese Liste nach der Signing Time. Wähle die jüngsten Kandidaten. Alle anderen werden ausgeschieden.
5. Gibt es mehrere, so wähle den besten Kandidaten:
    a. Prüfe auf Vertikale Länge:
        i. Wähle jene aus, welche die größte Anzahl an Feldern haben. Alle anderen werden ausgeschieden.
    b. Gibt es mehrere, so prüfe auf Horizontale Breite:
        i. Wähle jene aus, für die gilt, dass alle Captions jeweils länger oder gleich lang sein müssen als die entsprechenden Captions der anderen Kandidaten. Alle anderen werden ausgeschieden.
        ii. Werden im Verlauf dieser Untersuchung zwei Kandidaten entdeckt, für die gilt, dass zwar eine Caption des ersten Kandidaten länger ist als die entsprechende Caption eines anderen Kandidaten, dann aber eine andere Caption des anderen Kandidaten widerum länger ist als die entsprechende Caption des ersten Kandidaten, so wird ein Fehler generiert, weil in diesem Fall nicht bestimmt werden kann, welcher Kandidat der best geeignete ist.

   c. Gibt es mehrere, so muss für diese nun gelten, dass sie in Anzahl der Felder und Längen aller Captions gleich sind. Daher wird ein beliebiger dieser Kandidaten herangezogen.
6. Für den besten Kandidaten:
   a. Entferne den Text des Signaturblocks aus dem gegebenen Text.
   b. Restauriere den übrig gebliebene Text (entferne eventuell überschüssige newlines, etc.).
   c. Dieser Text ist jener, welcher für die Prüfung der Signatur herangezogen wird.
   d. Gehe Zu Schritt 2 und wiederhole den Algorithmus für diesen Text erneut so lange bis keine Signatur mehr gefunden wird.

Hinweis: Die Suche nach alten Signaturen wird wieder für den zum Schluss übrig gebliebenen Text durchgeführt. Annahme: Alte Signaturen werden beim Suchen nach neuen Signaturen nicht gefunden, weil ihnen das für neue Signaturen required SIG_KZ Feld fehlt.

Annahme: Der Wert der EndCaption befindet sich (durch das Normalisieren) im Text in einer Zeile. Damit kann man eindeutig das Ende eines Signaturblocks feststellen.

Annahme: Der User ist dafür verantwortlich, dass sich bei der Absoluten Positionierung der Text des Signaturblocks nicht mit dem Dokumenttext vermischt (verzahnt).

Hinweis: Der Knackpunkt dieses Algorithmus ist, dass das Normalisieren des Textes nach dem herausschneiden des Signaturblocks wirklich wieder den Originaltext ergibt.

Algorithmus „End Textsignatur“

Gegeben sei ein (bereits normalisierter) Text. Es sollen nun alle Signaturblöcke gefunden werden, wobei angenommen wird, dass die Signaturblöcke nicht absolut positioniert wurden und sich laut Signaturspezifikation daher am Ende des signierten Texts befinden.

1. Für alle definierten Signaturprofile:
   a. Von hinten nach vorne werden die Captions des Profils in Lesereihenfolge gesucht.
   b. Werden alle Captions im Text gefunden, so gilt das Profil als entdeckt.
   c. Werden mehrere Profile entdeckt, so gelten nur die Letztstehenden als entdeckt.
   d. Gibt es mehrere Letztstehende, so gelten nur die Kleinsten als entdeckt.
   e. Gibt es mehrere Kleinste, so muss unterschieden werden:
      i. Ob alle semantisch gleich lauten (ihre required Keys und Captions ident sind). In diesem Fall gilt das erste als entdeckt.
      ii. Ob die entdeckten nicht semantisch gleich lauten. In diesem Fall bricht der Algorithmus ab und gibt einen Fehler zurück.
   f. Ein entdecktes Profil wird abgetrennt und die Suche wird mit auf dem restlichen Text erneut ausgeführt.

Die Suche nach alten Signaturen verhält sich im Wesentlichen gleich, mit dem Unterschied, dass das SIG_KZ Feld für alte Signaturen ignoriert wird.

Normalisierungsmodul

Das Normalisierungsmodul hat die Aufgabe einen Roh Text anhand bestimmter Kriterien zu normalisieren. Die Normalisierungs- Kriterien werden in separaten Klassen definiert und umgesetzt. So können verschiedene Normalisierer Versionen erstellt werden. Diese werden über eine Factory zur Laufzeit geladen. D.h. beim Initialisieren des Normalisierungsmoduls kann auch die Version der Normalisierung mit angegeben werden. Wird dies explizit nicht angegeben, wird die default Einstellung aus der Konfigurationsdatei genommen.

```
normalizer.version=V01
```

Eine Normalisierungsklasse muss entsprechend das Normalize Interface implementieren. Die wichtigsten Methoden sind:

```
java.lang.String ← getNormCR()
java.lang.String ← normalize(java.lang.String rawText)
```

Die Normalizer Klasse verfügt zusätzlich noch über weitere Methoden, auf die aber an dieser Stelle nicht explizit eingegangen wird. Die Dokumentation dafür ist in der Java-Docu nachzulesen.

Die Normalisierungsschritte in der Version V01 sind der PDF-AS Amtssignaturen Spezifikation zu entnehmen.

Das nachfolgende Listing gibt eine Möglichkeit vor, das Normalisierungsmodul anzusprechen.

```
1  public static void main(String[] args) {
2    Normalizer norm = null;
3    try {
4      norm = new Normalizer();
5      String raw_string = "line 1: the \t raw string      \u0128Ä  \n\n";
6      raw_string += "line2 \u0060\u00B4\u2018\u2019\u201A\u201B \r\n";
7      raw_string += "line3 \u201C\u201D\u201E\u201F \r\n\r\n\r\r\r\r";
8      raw_string += "line4 \u00AD\u2013\u2014";
9      norm.setRawString(raw_string);
10     System.out.println(raw_string);
11     System.out.println(norm.getNormalizedString());
12   } catch (NormalizeException e) {
13     e.toString();
14   }
15 }
```

Erstellungsmodul

Die PDF-Repräsentation der Signatur Tabelle wird unter Zuhilfenahme der eingesetzten PDF-Library erzeugt.

Die Definition des Abstraktion Signatur Blocks und die Definition des Layouts der Signatur Tabelle ist ausführlich im Abschnitt 2 beschrieben (2.2.5 – 2.2.7).

Die eigentliche Tätigkeit des Erstellungsmoduls ist es nun zu erkennen, an welche Stelle die PDF Signatur Tabelle in das bestehende PDF-Dokument geschrieben werden muss, wenn dies nicht bereits absolut Spezifiziert wird. Dazu wird vom Extraktionsmodul die Position des letzten

Zeichens oder Bildes der letzten Seite angefordert. Ist nun noch genügend Platz für die PDF Signatur Tabelle vorhanden, wird diese auf der letzten Seite eingefügt. Ist der Platz dafür jedoch nicht mehr ausreichend, wird eine neue Seite erzeugt und die PDF Signatur Tabelle auf diese neue Seite geschrieben.

Bekannte Einschränkungen der vorliegenden Version

1. Die momentan eingesetzte PDF-Extraktions-Library ist in der Lage nur Zeichen korrekt auszulesen, die auch mit Acrobat Reader extrahiert werden können. Dies zeigen einerseits diverse Tests und dies stützt sich auch auf die Aussage der Enwickler besagter Library.
2. Fußnoten werden im Fliestext mit aufgenommen und stehen an der ersten Stelle einer Seite. Das ist auch bei Acrobat Reader so. Der Grund dafür dürfte in der Art und Weise liegen, wie der Dokumenttext in PDF erstellt wird.
3. Absolut positionierte Textsignaturen scheinen nicht „im Fließtext“ auf, sondern am Ende der jeweiligen Seite. Dies dürfte eine weitere Einschränkung der Text-Extraktions Library sein.
4. Seiten Nummern von PDF Dokumenten werden als Text erkannt, diese kommen ebenfalls in den Fliestext.
5. Texte in Grafiken z.B. Sprechblasen werden als Text extrahiert und in den Dokumenttext mit aufgenommen. Ist der Text allerdings Teil der Grafik, so wird dieser Text nicht erkannt.
6. PDF Zeichenobjekte (Linien, Rechtecke, Polygone, Bezier Kurven, etc.) werden weder bei der Textextraktion, noch beim Bestimmen der Seitenlänge berücksichtigt.