TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

Sicherheit und dynamische Veränderung von .pdf Dokumenten

Thema

ausgeführt am Institut für

E188 - Institut für Softwaretechnik und Interaktive Systeme

der Technischen Universität Wien

unter der Anleitung von

O.Univ.Prof. Dipl.-Ing. Dr.techn. A Min Tjoa und Univ.Ass. Dipl.-Ing. Dr.techn. Mag.rer.soc.oec. Edgar Weippl

(Name des Betreuers und gegebenenfalls auch des dabei verantwortlich mitwirkenden Universitätsassistenten, z.B. Prof.Dr. N. N. und Dipl.-Ing. N. N. als verantwortlich mitwirkendem Universitätsassistenten)

durch

Christoph Peinthor

Name

Leopoldauerstrasse 14/13

1210 Wien

Anschrift

Datum

Unterschrift (Student)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

<Ort, Datum> <Christoph Peinthor>

1. Inhaltsverzeichnis

2. Abstract

In unserem digitalen Zeitalter ist es zur Gewohnheit geworden, Dokumente nicht mehr als Brief zu senden, sondern dies digital per E-Mail zu machen.

Ein mittlerweile gängiges Mittel zur Erstellung von Formularen oder zum Versand von Briefen sind .pdf Dokumente. Mit .pdf Dokumenten ist es sowohl möglich Text darzustellen als auch interaktive Elemente für den Benutzer in das Dokument einzubauen.

Diese Art von Dokument hat sich weit verbreitet, da es freie Drucker (Programme) zur Erstellung von .pdf Dokumenten gibt und mit dem Adobe Reader ein hochwertiges Programm zur Betrachtung e zur Verfügung steht.

Doch wer garantiert, dass das am Bildschirm gesehene auch das originale Dokument ist?

Acrobat Adobe stellt eine Javascript-Schnittstelle zur Verfügung, die es dem Benutzer erleichtern soll Dokumente auszufüllen. Doch mit dieser Technologie lassen sich Dokumente nicht nur verschönern, sondern auch zur Laufzeit verändern. Hierzu werden bestimmte Javascriptprogramme entweder auf Aktionen in dem Dokument, der Seite oder einem Formularfeld gelegt. Somit ist es möglich auf Benutzeraktionen zu reagieren und das Dokument zu verändern.

Ziel dieser Arbeit ist es, sowohl mit der in Acrobat Adobe integrierten Javascript-Schnittstelle, als auch durch Veränderung des .pdf Codes eines solchen Dokuments dem Benutzer einen anderen Inhalt zu zeigen, als dann letztendlich vom Benutzen bestätigt wird.

Weiters wird gezeigt wie es möglich ist, ohne das Dokument zu öffnen, Javascriptvorgänge einzubinden und somit das Dokument ohne das Wissen des Autors zu verändern, was durch eine Historisierung beim Speichern eines .pdf Dokuments in herkömmlicher Form nicht durchführbar ist.

Da das Ändern eines Dokuments zu Laufzeit durch den Ersteller eingebaut wird, kann hier außer bei einem signierten Dokument nie von einem unveränderten Dokument ausgegangen werden. Die Veränderung durch dritte, sowohl in dem Dokument als auch in dem .pdf Code, ist nur bei ungesicherten Dokumenten möglich.

3.Einleitung

3.1. Was ist ein pdf Dokument und Adobe

Das Format PDF wurde von der Firma Adobe entwickelt und basiert auf der Sprache Postscript. Bei dieser Sprache handelt es sich um eine Seitenbeschreibungssprache.

Die wichtigste Eigenschaft eines .pdf Dokuments ist, dass es die Schriften, Zeichnungen und Farben des Ursprungsdokuments so genau wie möglich wiedergibt, wobei der Inhalt für den nächsten Bearbeiter nur zum Lesen und nicht zum Editieren gedacht ist. Weiters kann das Dokument noch mit 40 oder 128 Bit verschlüsselt werden, um sicher zu gehen, dass es keine Veränderungen gibt. Zu einer Verschlüsselung kommt es bei jeder Art von Sicherung des Dokuments, egal ob es sich um eine generelle Sicherung handelt, welche das Öffnen des Dokuments verhindert, oder nur um eine Verminderung der Rechte auf ein Dokument. Nur durch eine solche Sicherung und damit auch Verschlüsselung ist die Möglichkeit unterbunden, den Code des Dokuments direkt zu verändern.

Den Durchbruch hat das PDF Format durch die Entstehung von vielen PDFViewern[1] gemacht, mit denen es möglich wurde .pdf Dokumente in jedem Betriebssystem zu betrachten und auch zu drucken.

Erst als nächste Stufe kam es zu der Entwicklung von PDF Designern, mit welchen nicht nur das Ansehen eines zuvor aus Postscript erstellten Dokuments möglich war, sondern auch das Verändern dieses Dokuments.

Eine Untersuchung im Jahre 2005 hat ergeben, dass bei einer Anfrage in Google ca. 10% der Ergebnisse ein .pdf Dokument liefern. Dies Ergebnis zeigt die weite Verbreitung des .pdf Formates im World Wide Web. [22]

3.1.1. Schützen vor Veränderung

Nachdem die Veränderung von .pdf Dokumenten denkbar wurde, sind für .pdf Dokumente auch Sicherheitslücken entstanden, da es jedem leicht durchführbar war ein .pdf Dokument zu verändern.

Deshalb ist es möglich beim Druck oder auch bei der Speicherung von .pdf Dokumenten anderen Benutzers mittels einer Passwortvergabe gewisse Zugriffe oder Aktionen in einem .pdf Dokument zu untersagen.

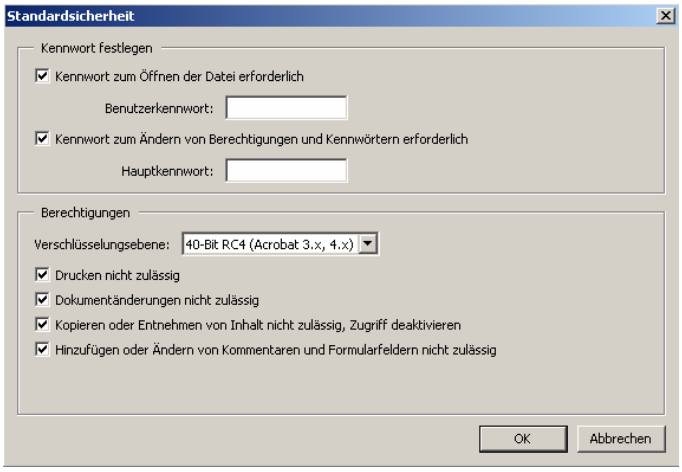


Abbildung 1: Kennwortschutz

[1] PdfViewer sind Programme mit denen sich .pdf Dokumente ansehen lassen. Jedoch ist eine Veränderung des Inhalts, nur mit Kommentaren möglich.

Es ist hier sowohl eine 40 sowie eine 128–Bit Verschlüsselung einsetzbar.

3.1.2. Skandale mit .pdf Dokumenten

Im Gegensatz zu anderen Dokumenten wird bei der Erstellung eines .pdf Dokuments sehr viel an Information mitgeschrieben. So ist es möglich durch das Öffnen eines .pdf in einem Editor viele Informationen zu finden. Hier steht z.B.:

- Erstellungsdatum
- Name der Person die dieses Dokument erstellt hat
- Womit dieses Dokument erstellt wurde (z.B.: Acrobat Distiller)
- Unter welchem Betriebssystem das Dokument erstellt wurde
- Wann das Dokument zuletzt verändert wurde
- Wer das Dokument zuletzt verändert hat
- Dateiname der Datei aus der das Dokument erstellt wurde

```
<?xpacket begin='' id='W5MOMpCehiHzreSzNTczkc9d' bytes='1329'?>

<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
 xmlns:iX='http://ns.adobe.com/iX/1.0/'>

 <rdf:Description about=''
  xmlns='http://ns.adobe.com/pdf/1.3/'
  xmlns:pdf='http://ns.adobe.com/pdf/1.3/'>
  <pdf:CreationDate>2006-11-04T10:11:58Z</pdf:CreationDate>
  <pdf:ModDate>2006-11-11T12:07:50+01:00</pdf:ModDate>
  <pdf:Producer>Acrobat Distiller 5.0 (Windows)</pdf:Producer>
  <pdf:Author>Christoph</pdf:Author>
  <pdf:Creator>PScript5.dll Version 5.2.2</pdf:Creator>
  <pdf:Title>mein PDF - Editor</pdf:Title>
 </rdf:Description>

 <rdf:Description about=''
  xmlns='http://ns.adobe.com/xap/1.0/'
  xmlns:xap='http://ns.adobe.com/xap/1.0/'>
  <xap:CreateDate>2006-11-04T10:11:58Z</xap:CreateDate>
  <xap:ModifyDate>2006-11-11T12:07:50+01:00</xap:ModifyDate>
  <xap:Author>Christoph</xap:Author>
  <xap:MetadataDate>2006-11-11T12:07:50+01:00</xap:MetadataDate>
  <xap:Title>
   <rdf:Alt>
    <rdf:li xml:lang='x-default'>mein PDF - Editor<
```

Abbildung 2: Informationen über .pdf in XML

Durch die Speicherung solcher Daten kann es zu ungewollten Veröffentlichungen kommen wie das folgende Beispiel zeigt (nähere Beschreibung siehe Abschnitt 8.3):

Das Weiße Haus in Washington (D.C.) veröffentlichte George W. Bushs Rede zum „Plan für den Sieg im Irak". Die Dateiangaben legten den Ghostwriter offen, nämlich Peter Feaver, Professor für Politikwissenschaften von der Duke Universität in North Carolina, welcher seit Juni 2005 das National Security Council berät [1].

Wie in weiterer Folge beschrieben, ist es nicht möglich, über Acrobat Adobe den Ersteller oder einen Editor des Dokuments zu verändern. Jedoch ist es mit

ausreichendem Wissen über den .pdf Code nicht schwierig in dem Code die Passagen mit den Benutzerinformationen zu finden, und diese direkt in dem .pdf Code zu verändern.

3.1.3. Szenarien

Dieses Dokument beschreibt außerdem Angriffs- und Betrugszenarien welche mit einem .pdf Dokument möglich sind. Im Speziellen werden 2 Szenarien beschrieben:

- Veränderung durch den Ersteller: In diesem Fall befinden sich in einem Dokument Stellen, die durch den Ersteller eingefügt wurden, und einem anderen Benutzer nicht bekannt sind.

Dabei hat der Ersteller alle Rechte in dem Dokument.

- Veränderung durch andere: In diesem Fall handelt es sich um ein Dokument, welches von einer Person, welche nicht autorisiert ist verändert wird. Dabei kann es sich um ein Dokument handeln, welches ungeschützt oder geschützt ist.

Die Veränderung kann in dem Dokument erfolgen, oder auch in dem Code des .pdf Dokuments.

3.2. Javascript

In diesem Kapitel werden Javascriptbefehle und Datentypen welche im Weiteren in diesem Dokument verwendet werden beschrieben. Es wird auch ein kurzer allgemeiner Überblick über die Entstehung und die Geschichte von Javascript gegeben.

3.2.1. Was ist Javascript

Javascript ist eine Programmiersprache, welche zum Unterschied von serverseitigen Programmiersprachen wie Perl, PHP oder C# vom Client interpretiert wird. Der Client an sich stellt jedoch nur die Rechenleistung zur Verfügung, denn das eigentliche Interpretieren der Sprachen übernimmt der Webbrowser.

Aufgrund dieser Eigenschaft ist Javascript unabhängig vom Betriebssystem. Es kann jedoch nicht jeder Webbrowser alle Javascriptbefehle interpretieren. Weiters ist Javascript nicht mit der namensähnlichen Programmiersprache Java verwandt.

Javascript kann entweder direkt in ein HTML Dokument eingebunden werden indem es in den Tag <script></script>[2] eingebunden wird, oder in einer externen Datei platziert werden, welche meist die Endung .js aufweist.

Javascript wurde 1995 zum ersten Mal in den von Netscape veröffentlichen Navigator 2.0 eingebaut, um Formulareingaben vor dem Absenden zu einem Server zu überprüfen.

„JavaScript is a scripting language designed for enhancing web pages. JavaScript programs are deployed in HTML documents and are interpreted by all major web browsers. They provide useful client-side computation facilities and access to the client system, making web pages more engaging, interactive, and responsive."[19]

In diesem Fall wird Javascript jedoch nicht von einem Browser interpretiert, sondern von Acrobat. Hierbei ist nicht wichtig ob die volle Version des Adobe Acrobat oder nur der „Viewer" Acrobat Reader verwendet wird.

Da es sich hier um Javascript handelt, welches nicht von einem Standardbrowser interpretiert wird, kommt es auch zu einer leichten Änderung der Syntax im Vergleich zu allgemein bekannten Javascript.

3.2.2. Verwendete Javascriptbefehle und Datentypen

In den folgenden Beispielen werden standardisierte Javascriptbefehle und Methoden verwendet. Im Folgenden sind diese ausführlich beschrieben

Datentyp	Beschreibung
String	Dieser Typ beschreibt eine Variable welche lediglich Text speichern kann. Es ist zwar möglich auch eine Zahl als Text zu speichern, jedoch kann bei diesem Datentyp auch bei numerischen Werten kein Vergleich auf größer oder kleiner gemacht werden. Auch mathematische Operationen werden nicht unterstützt.
Int	Dieser Datentyp speichert nur ganzzahlige Werte. Dezimalzahlen werden auch nicht gerundet, sondern „abgeschnitten"(der Teil nach dem Komma wird weggelassen).
Boolean	Bei diesem Datentyp handelt es sich um einen logischen Datentyp. Es kann lediglich der Wert `true` und `false` gespeichert werden.

Tabelle 1: Javascript Datentypen [3]

Befehl	Erklärung
var	Mit diesem Operator wird eine neue Variable definiert. `Var neueVariable = „neu"` Somit ist der Variablen `neueVariable` der Wert `neu` zugewiesen.

toString	Die Methode toString() formatiert einen Wert einer Variablen zu einem String.
	`Var Zahl = 2`
	`var Text = Zahl.toString()`
	In diesem Beispiel wird eine Variable `Zahl` definiert und danach der Variablen `Text` der Wert der Variablen `Zahl` nicht als numerischer Wert sondern als normaler Text zugewiesen

parseInt(`Zahl`)	Es wird die Variable, sofern dies möglich ist, in eine ganze Zahl umgewandelt und es stehen alle mathematischen Operatoren zur Verfügung
variable.value	Es wird mittels „value" auf den Wert der

	angegebenen Variablen zurückgegriffen.
IF(Bedingung) {CODE}	Bei dieser Funktion wird auf eine bestimmte Bedingung reagiert. Ist diese Bedingung erfüllt, So wird der nachfolgende Code ausgeführt. Wird die Bedingung nicht erfüllt, wird der Codeteil übersprungen.
IF(Bedingung){CODE} ELSE {CODE}	Hier wird auf eine bestimmte Bedingung reagiert. Ist diese Bedingung erfüllt, so wird der nachfolgende Code ausgeführt. Wird die Bedingung nicht erfüllt, wird der Codeteil welcher sich nach dem „else" befindet ausgeführt.
Util.print	Dies wird verwendet um aus einem übergebenen Datum einen bestimmten Bereich (Tag, Monat, Jahr) auszulesen.

Tabelle 2: Javascript Befehle [28]

4. Erstellung und Bearbeitung von .pdf Dokumenten

In diesem Kapitel wird die Erstellung eines .pdf Dokuments mit Acrobat Adobe 5.0 Distiller beschrieben. Alle Benutzereingaben in Javascript sind auch in der Version 7.0 und 8.0 in gleicher Weise funktionstüchtig. Es unterscheiden sich lediglich die Orte im Menü in denen sich die Tools befinden.

4.1. Erstellung eines .pdf Dokuments

Die einfachste Möglichkeit ein .pdf Dokument zu erstellen ist die Generierung eines.txt oder .doc Dokuments mit dem entsprechenden Text. Dieses Dokument kann dann mit dem **Acrobat Distiller** einfach als .pdf Dokument gedruckt werden.

Eine weitere Variante ist das Drucken eines .pdf Dokuments aus einem Postscript Dokument.

Der **Acrobat Distiller** ist ein Drucker der bei der Installation von **Adobe Acrobat** automatisch installiert wird, und bei den Druckoptionen als Drucker ausgewählt werden kann. Bei der Installation von 7.0 wir auch ein eigner Menüeintrag in Microsoft Word gemacht um den Acrobat Distiller direkt ansprechen und .pdf Dokumente erzeugen zu können. Danach kann der Zielort und der Name des neuen .pdf Dokuments ausgewählt werden.

Weiters kann der Distiller auch als Programm aufgerufen werden, um Voreinstellungen zu treffen. Es ist somit auch möglich in dem Acrobat Distiller dieselben Sicherheitseinstellungen wie unter Abschnitt 9.1 beschrieben als Standard zu verwenden und auf jeden Druck anzuwenden.

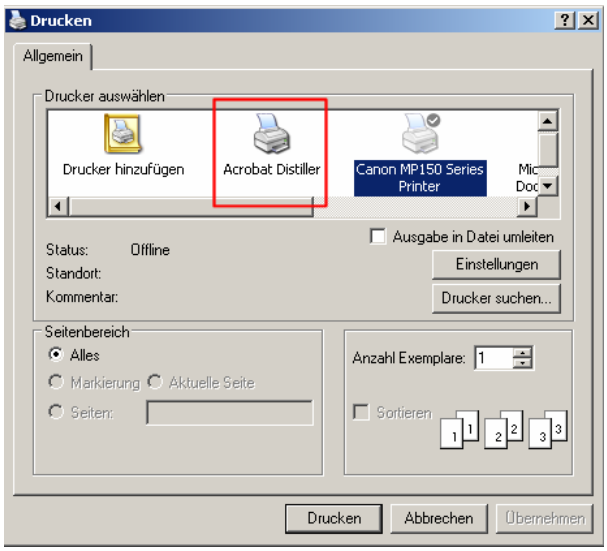


Abbildung 3: Acrobat Distiller

Danach kann das erzeugte .pdf Dokument geöffnet und bearbeitet werden.

4.2. Grundeinstellungen in Adobe Acrobat

Die folgenden Einstellungen müssen aktiviert sein, damit Javascript von **Adobe Acrobat** interpretiert werden kann. Bei einer Standardinstallation sind diese Optionen bereits aktiviert. Um sich jedoch vor ungewollten Veränderungen in .pdf Dokumenten zu schützen, empfiehlt es sich diese Einstellung nach dem Arbeiten mit Javascript wieder zu deaktivieren.

Viele Formulare verwenden auch die Javascriptfunktionen, um dem Bearbeiter das Bearbeiten eines Dokuments zu vereinfachen. Ohne Javascript ist es nicht möglich alle Funktionen eines .pdf Dokuments mit Formularen zu nutzen.[29]

Bearbeiten → Grundeinstellungen → Allgemein…

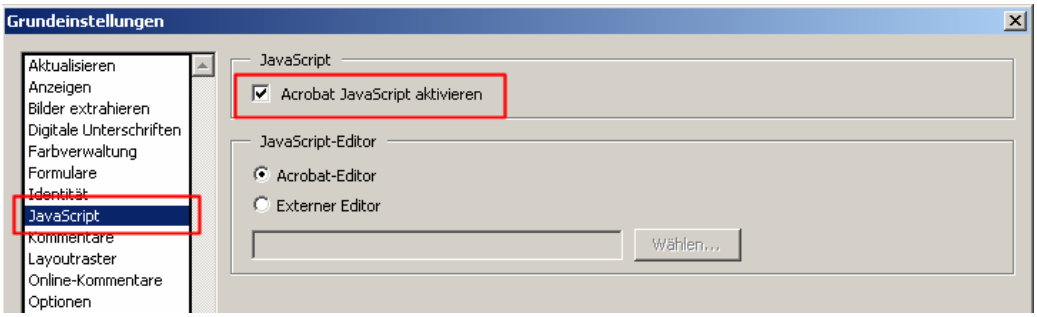


Abbildung 4: Aktivierung von JavaScript

4.3. Formularfelder

Formularfelder sind Objekte in einem .pdf Dokument welchen besondere Eigenschaften hinterlegt sind. Formularfelder können durch einen einfachen Dialog Berechnungen durchführen oder als Kontrollkästchen auftreten.

Es ist auch möglich durch Javascript, welches in Formularfelder eingebunden wird, eigenen Text oder komplexe Berechnungen unter Abhängigkeit von anderen Werten in Formularfeldern oder von Umgebungsvariablen wie Datum oder Speicherzahl durchzuführen.

Für die dynamische Veränderung von .pdf Dokumenten sind Formularfelder unerlässlich, da sie durch ihren Namen direkt angesprochen werden und so auch ihre Inhalte (Werte) ausgelesen und verwendet werden können.

4.3.1. Typen von Formularfeldern

Ein Formularfeld kann in einem .pdf Dokument die unterschiedlichsten Formen annehmen:

Name	Beschreibung	Aussehen
Kombinationsfeld	Ein Kombinationsfeld erlaubt dem Benutzer die Auswahl eines Wertes aus einer Reihe von vordefinierten Werten.	Element1 Element1 Element2 Element3
Kontrollkästchen	Ein Kontrollkästchen ist ein Objekt, dass lediglich die	False

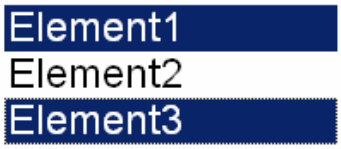
	Werte `true` und `false` in Form eines Häkchens anzeigt.	 `true` 
Listenfeld	Ein Listenfeld gibt dem Benutzer eine Liste von Werten zur Auswahl, aus welchen dieser einen (bei Konfiguration) auch mehrere Werte auswählen kann.	
Optionsfeld	Ein Optionsfeld kann nur den Wert `true` oder `false` haben und macht als Einzellobjekt wenig Sinn. Hier ist es erforderlich mehrere Optionsfelder mit demselben Namen zu definieren. Dadurch entsteht eine Objektgruppe, in der aber nur ein Objekt den Wert `true` annehmen kann.	Entweder…  …oder 
Schaltfläche	Eine Schaltfläche ist ein Objekt welches nur in Verbindung mit Vorgängen (Abschnitt 4.3.2.3) das Bearbeiten eines Dokument beeinflussen kann. Die Schaltfläche kann von einem Bearbeiter angeklickt werden.	
Text	In einem solchen Feld kann der Benutzer eigene Eingaben machen, sofern dies so konfiguriert ist.	
Unterschrift	Hier kann der Benutzer das Dokument digital signieren.	

Tabelle 3: Formularfelder Typen

4.3.2. Konfiguration von Formularfeldern

Nach dem Einfügen eines Formularfeldes muss dieses konfiguriert werden. Es werden hier nur die wichtigsten und für dieses Dokument auch relevanten Attribute der Formularfelder beschrieben. Nicht jeder Typ von Formularfeld hat auch dieselbe Anzahl an Tabs zur Konfiguration zur Verfügung.

Abbildung 5: Formularfeld Allgemein

- **Name**: Ein eindeutiger Name für dieses Katalogfeld wird vergeben. Wenn zwei Katalogfelder denselben Namen haben, werden die Inhalte bei Veränderung des einen Feldes auch in das andere Feld kopiert. Handelt es sich allerdings um Optionsfelder, so gilt hier eine Speziallogik. Bei einer Menge von Optionsfeldern mit gleichem Namen, kann immer nur eines dieser Optionsfelder markiert werden.
- **Typ**: Hier wird der Typ des Katalogfeldes wie in Abschnitt 4.3.1. beschrieben angegeben.
- **Kurze Beschreibung**: Optionales Textfeld für eine Benutzereingabe.

4.3.2.1. Darstellung

Auf diesem Tab wird das Aussehen für das Formularfeld bestimmt.

Abbildung 6: Formularfeld Darstellung

- **(Umrandung)Farbe**: Gibt die Farbe des Rahmens an
- **(Umrandung)Breite**: Gibt die Stärke des Rahmens an (Dieses Element kann im .pdf Code durch hinzufügen des Schlüssels /BS und der folgenden Werte eingestellt werden. Die Zahl „3" gibt den ausgewählten Wert in der Drop Down Box in Acrobat Adobe an << /W 3 /S /S >> Beispiel: „/BS << /W 3 /S /S >>")
- **(Umrandung)Hintergrund**: Gibt die Hintergrundfarbe des Formularfeldes an
- **(Umrandung)Art**: Gibt die Art des Rahmens an (Dieses Element kann im .pdf Code durch hinzufügen des Schlüssels /BS und der folgenden Werte eingestellt werden `/BS << /W 1 /S /D /D [ 3 ] >>.` Wobei folgende Schlüssel verwendet werden. /S = Durchgehend; /D = Unterbrochen; /B = Relief; /I = Eingerückt; /U = Unterstrichen)
- **(Textfeld)Farbe**: Gibt die Schriftfarbe an
- **(Textfeld)Schrift**: Gibt die Schriftart an (Dieses Element kann im .pdf Code durch hinzufügen des Schlüssels /DA und der folgenden Werte eingestellt werden `/DA (/Cour 12 Tf 0 g).` Der erste Wert in den Klammern gibt die Schriftart an, wobei jede Schrift ihre eigene Abkürzung hat)
- **(Textfeld)Größe**: Gibt die Schriftgröße an (Dieses Element kann im .pdf Code durch hinzufügen des Schlüssels /DA und der folgenden Werte eingestellt werden. `/DA (/Cour 12 Tf 0 g).` Der zweite Wert in den Klammern gibt die Schriftgröße an)

- **(All. Eigenschaften)Schreibgeschützt**: Auswahl ob das Feld editiert werden darf (Dieses Attribut kann nicht über den Code aktiviert oder deaktiviert werden)
- **Formularfeld ist**: Kann folgende Ausprägungen haben: (Diese Attribute können nicht über den Code aktiviert oder deaktiviert werden)
    - Sichtbar
    - Unsichtbar
    - Sichtbar, aber Drucken nicht möglich
    - Unsichtbar, aber Drucken möglich
- **Erforderlich**: Gibt an ob dieses Formularfeld leer sein darf(Dieses Attribut kann nicht über den Code aktiviert oder deaktiviert werden)

4.3.2.2. Optionen

Bei dem Tab „Optionen" handelt es sich um spezifische Eigenschaften eines Formularfeldes, wobei sich dieser Tab je nach Typ eines Formularfeldes ändert.

- **Kombinationsfeld, Listenfeld**

Abbildung 7: Formularfeld Optionen 1

- Element: Durch Eingabe eines Namens und des Buttons *Hinzufügen*, kann die Liste der Vorgaben befüllt werden.
- Exportwert: Der Wert der bei Auswahl dieses Elements exportiert wird.
- Elemente sortieren: Elemente nicht in der eingegebenen Reihenfolge, sondern alphabetisch sortieren.

- **Kontrollkästchen**

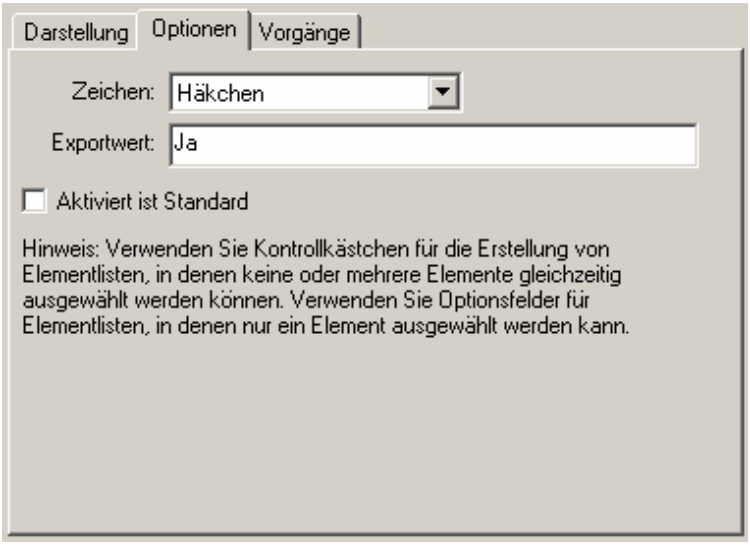


Abbildung 8: Formularfeld Optionen 2

- Zeichen: Mit diesem Zeichen wird das Kontrollkästchen bei Auswahl befüllt.
- Exportwert: Der Exportwert bei aktiviertem Kontrollkästchen.
- Aktiviert ist Standard: Auswahl ob aktiviert oder nicht.

4.3.2.3. Vorgänge

Dieser Tab steht für alle oben genannten Felder zur Verfügung.

Bei Vorgängen, handelt sich schon um eine Art dynamische Veränderung eines .pdf Dokuments. Es wird auf eine bestimmte Aktion im Bezug auf ein Formularfeld ein Event[2] gelegt.

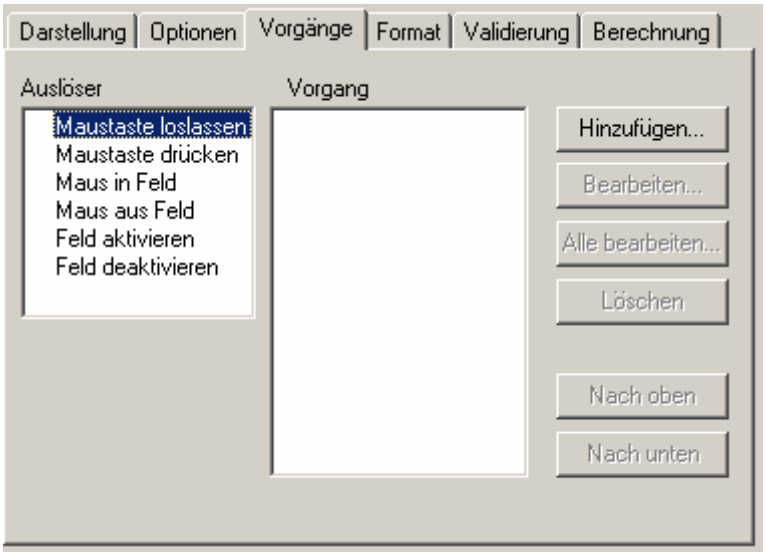


Abbildung 9: Formularfeld Optionen 3

[2] Event: Vom Benutzer oder vom System ausgelöstes Ereignis.

Auf der linken Seite der Tabelle 4 ist in dem Feld „Auslöser" eine Liste von Events aufgelistet, auf welche ein Vorgang gesetzt werden kann.

Auslöser	Beschreibung
Maustaste loslassen	Wenn sich der Mauszeiger über dem Formularfeld befindet und losgelassen wird.
Maustaste drücken	Wenn sich der Mauszeiger über dem Formularfeld befindet und gedrückt wird.
Maus in Feld	Hier wird ein Vorgang ausgelöst, sobald sich der Mauszeiger über das Formularfeld bewegt.
Maus aus Feld	Hier wird ein Vorgang ausgelöst, sobald sich der Mauszeiger aus dem Formularfeld hinaus bewegt.
Feld aktivieren	Hier wird ein Vorgang ausgelöst, sobald z.B. ein Kontrollkästchen aktiviert (angehakt) wird, jedoch nicht wenn ein aktives Formularfeld deaktiviert wird.
Feld deaktivieren	Hier wird ein Vorgang ausgelöst, sobald z.B. ein Kontrollkästchen deaktiviert (nicht angehakt) wird, jedoch nicht wenn ein deaktiviertes Formularfeld aktiviert wird.

Tabelle 4: Formularfelder Auslöser

4.4. Zusammenfassung

Nach der Erstellung eins .pdf Dokuments kann dieses sehr einfach nachbearbeitet werden. Es kann Text hinzugefügt, entfernt oder auch formatiert werden.

Zusätzlich können Formularfelder in einem Dokument auch bestimmte Eigenschaften haben.

Die wichtigsten Eigenschaften sind ein eindeutiger Name sowie die Möglichkeit Vorgänge auf Auslöser zu hängen. Auslöser sind bestimmte Zustände des Formularfeldes, wie das Drücken oder das Loslassen einer Schaltfläche.

Weiters können die Formularfelder auch formatiert werden, oder bestimmte Spezialaufgaben haben.

5. Dynamische Veränderung von .pdf Dokumenten

In diesem Kapitel werden die Möglichkeiten beschrieben die es gibt um ein .pdf Dokument zu verändern.

Es werden als erstes Vorgänge beschrieben, mit denen sich unter anderem auf Aktionen in Formularfeldern reagieren lässt. Diese Anwendung ist relativ einfach da es sich großteils um vorgefertigte Dialoge handelt die durch „klicken" bearbeitet werden können und vom Benutzer keine Programmierkenntnisse verlangt werden.

Im Weiteren wird mit einem „Event Handler" auf andere Aktionen in einem .pdf Dokument reagiert und ein Javascript ausgeführt.

5.1. Vorgänge

Vorgänge sind einfache Aktionen die von einem Benutzer mittels eines Dialoges in einem .pdf Dokument erstellt werden können. Ein einfaches Beispiel für einen Vorgang wäre das Öffnen des Standardbrowser mit einer bestimmten URL.

Jeder Vorgang muss jedoch an ein bestimmtes Event gebunden sein damit er bei Eintreten dieses Events auch ausgeführt werden kann.

Mit vielen Vorgängen kann ein .pdf Dokument für den Benutzer grafisch aufgebessert, oder die Benutzerfreundlichkeit mit Verlinkungen erhöht werden. Es gibt jedoch auch Vorgänge mit denen ein .pdf Dokument dynamisch verändert wird.

5.1.1. Hinzufügen von Vorgängen

Es soll ein Vorgang zu einem Formularfeld hinzugefügt werden der auf das .pdf Dokument Auswirkungen hat.

5.1.1.1. Feld ein-/ausblenden

Durch diesen Vorgang hat ein Feld oder dessen Status Auswirkungen auf ein anderes Feld.

In dem folgenden Szenario soll realisiert werden, dass ein Textfeld durch eine Schaltfläche ein- und ausgeblendet werden kann.

1. Hinzufügen von 3 Formularfeldern zu dem .pdf Dokument. Dabei handelt es sich um ein Feld des Typs Textfeld mit dem Namen `Text1`. Bei den anderen beiden Feldern um Schaltflächen mit den Namen `Sfein` und `Sfaus`.

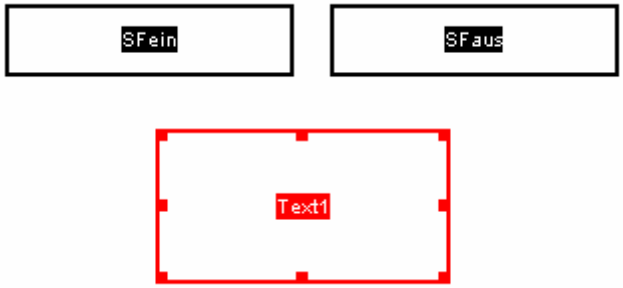


Abbildung 10: Vorgang Szenario 1, Aufbau

2. Nun wird das Eigenschaftenfenster des Feldes `Sfein` geöffnet und zu dem Tab Vorgänge gewechselt.

3. Auf diesem Tab wird ein neuer Vorgang auf den Auslöser „Maustaste drücken" eingefügt.

4. Es wird der Vorgang „Feld ein-/ausblenden eingefügt.

Abbildung 11: Vorgang Szenario 1, Vorgangsauswahl

5. Nun wird der ausgewählte Vorgang noch mittels der Schaltfläche „Bearbeiten..." konfiguriert.

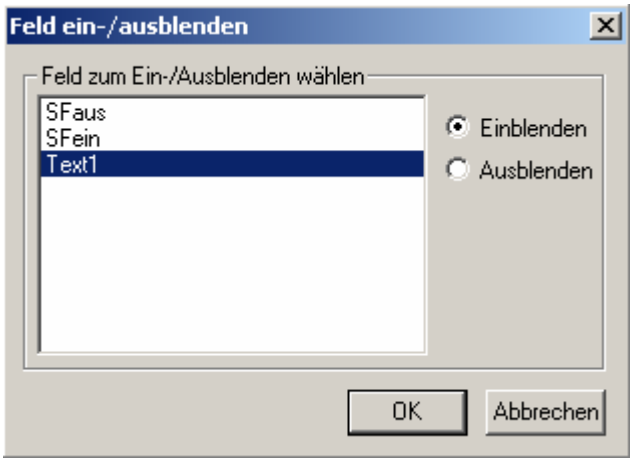


Abbildung 12: Vorgang Szenario 1, Vorgangsbearbeitung

Hier ist das Formularfeld Text1 zu wählen und die Option „Einblenden".

6. Alle Eingaben bestätigen und zur Ausgangsansicht zurückkehren.

7. Nun wird noch das Formularfeld `Sfaus` konfiguriert. Dies wird durch das Ausführen der Schritte 2 – 5 realisiert. Jedoch wird hier beim Bearbeiten des Vorganges (Abbildung 12) die Option „Ausblenden" gewählt.

Nach der Durchführung der Schritte 1 – 7 kann mittels einer Schaltfläche ein anderes Formularfeld ein- oder ausgeblendet werden.

5.1.1.2. Formulardaten senden

Mit diesem Vorgang wird ein .pdf Dokument welches von einem Benutzer in einem Webbrowser geöffnet wurde versendet. Dieses Dokument wird nicht per Mail versendet sondern direkt an die angegeben URL in einem vordefinierten Format.

Dies empfiehlt sich, wenn der Benutzer z.B.: einen Bestellschein ausgefüllt hat um diesen sofort an den Empfänger zu senden.

5.1.1.3. Formulardaten zurücksetzen

Diese Option setzt ein verändertes Formularfeld wieder auf den Ausgangswert.

5.1.1.4. Menübefehl ausführen

Mit diesem Vorgang kann jeder Menübefehl ausgewählt und auch ausgeführt werden. So ist es zum Beispiel möglich mit einer Schaltfläche die Hilfe des Adobe Acrobat aufzurufen.

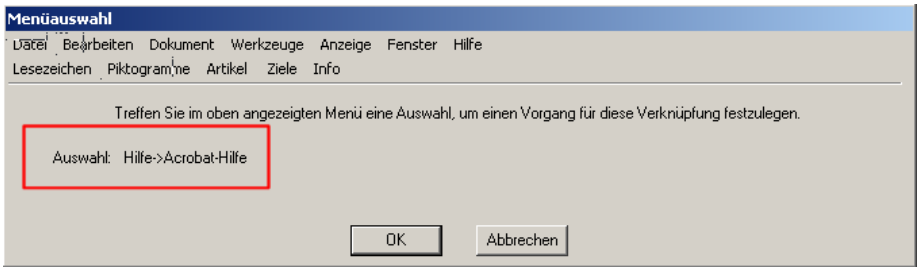


Abbildung 13:Menübefehl ausführen

5.1.1.5. JavaScript

Mit diesem Aufruf wird ein benutzerdefiniertes Javascript ausgeführt.

Durch die Möglichkeit diesen Vorgang aufzurufen, können an Formularfelder beliebig viele Aktionen gebunden werden.

5.1.2. Kombinieren von Vorgängen

Wenn zu einem Formularfeld mehr als ein Vorgang hinzugefügt wurde, so kann man den Ablauf der Vorgänge zu beeinflussen.

Die aufgelisteten Vorgänge werden immer von oben nach unten abgearbeitet, wobei die Position eines Vorganges in der Liste der Vorgänge vom Benutzer gewählt wird.

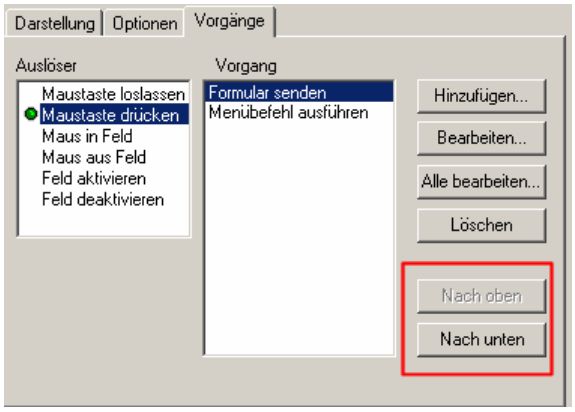


Abbildung 14: Verschieben von Vorgängen

5.1.2.1. Szenario mit mehreren Vorgängen

Nachfolgend wird ein Szenario beschrieben in dem mehrer Vorgänge hintereinander ausgeführt werden.

Das Ziel dieses Szenarios ist es, ein Produkt zu bestellen und dann mit einer „Senden" Schaltfläche zu versenden und sofort das Dialogfenster „Drucken…" zu öffnen. Jedoch wird davor noch ein Feld auf den Ausgangsstatus zurückgesetzt.

Hierbei handelt es sich um ein Kontrollkästchen das dem Benutzer erlaubt eine überhöhte Transportversicherung abzuschließen. Es wir angenommen, dass der Benutzer das nicht beabsichtigt und die Option nicht auswählt.

Durch den Vorgang „Formulardaten zurücksetzen" wird jedoch der Standardwert (angehakt) in dieses Feld geschrieben. Da dieser Vorgang als erstes ausgeführt wird, wird die Bestellung sowohl mit dieser Option versendet als auch für den Benutzer gedruckt.

1. Das Bestellformular kann wie folgt aussehen, wobei hier angenommen wird, dass der Kunde die Transportversicherung nicht wünscht.

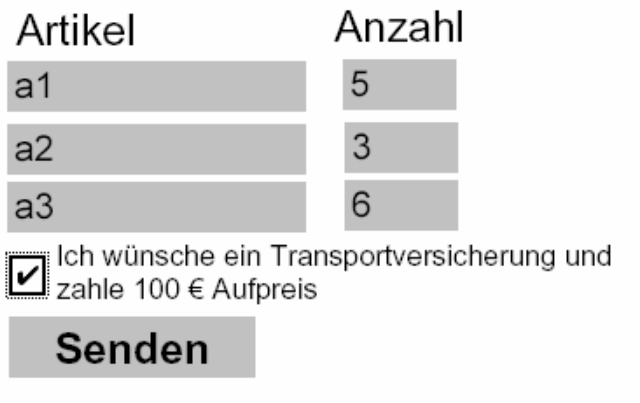


Abbildung 15: Vorgang Szenario 2, Grundformular

2. Konfiguration des Kontrollkästchens: Hier ist es wichtig in den Eigenschaften des Formularfeldes unter dem Tab „Optionen" das Kästchen „Aktiviert ist Standard" zu setzen.

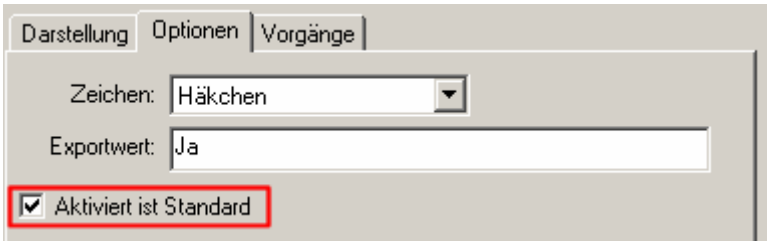


Abbildung 16: Vorgang Szenario 2, Kontrollkästchen Eigenschaften

3. Konfiguration der Vorgänge der Schaltfläche „Senden" auf den Auslöser „Maustaste drücken":

   a. Formular zurücksetzen: Hier kann mittels Klicken auf die Schaltfläche „Felder auswählen…" das gewünschte Feld ausgewählt werden.

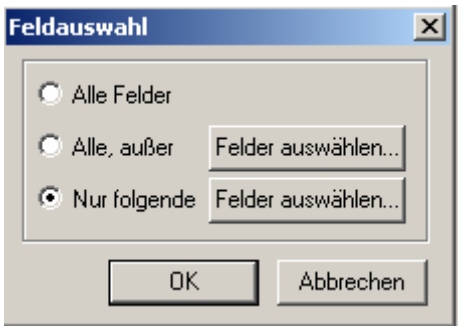


Abbildung 17: Vorgang Szenario 2, Feldauswahl

Hier wird die Option „Nur folgende" ausgewählt, und diese mit der Schaltfläche „Felder auswählen" konfiguriert.

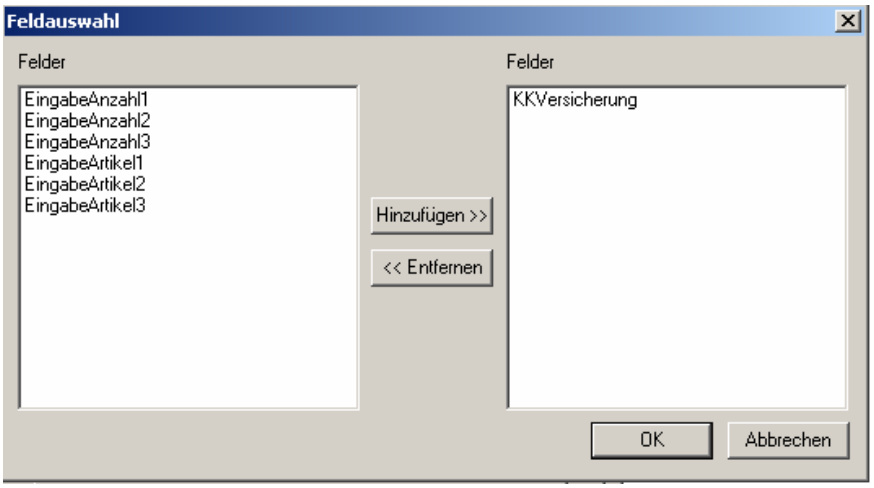


Abbildung 18: Vorgang Szenario 2, Kontrollkästchen auswählen

Somit wird nur dieses Feld zu dem Standardverhalten zurückgesetzt.

b. Formular senden: Hier muss die Konfiguration für das Senden des Formulars eingegeben werden.

c. Menübefehl ausführen: In diesem Punkt wird der Menubefehl (genaueres siehe Abschnitt: 5.1.1.4) für das Öffnen des Druckdialoges konfiguriert.

ANMERKUNG: Das oben beschriebene Szenario ist Betrug, da der Benutzer gegen seinen Wunsch die Versicherung zahlen muss. Dies ist sowohl in dem gesendeten Formular als auch auf dem Ausdruck so festgehalten.

5.1.3. Grenzen von Vorgängen

Mittels der Bearbeitung von Vorgängen kann vieles in einem .pdf Dokument verändert werden. Jedoch wird bei Vorgängen das Formularfeld als Ganzes betrachtet, wodurch sich keine Differenzierungen auf Werte in Formularfeldern herausarbeiten lassen.

Da Vorgänge wie schon beschrieben nicht auf Werte in Formularfeldern Bezug nehmen, ist zum Beispiel die einfache Aufgabe eines Kontrollkästchens, durch welches je nach Wert ein Textfeld freigeschalten wird, nicht über Vorgänge abbildbar.

Die einzige Alternative um wertbezogen eine Aktion auszuführen, ist diese in einen Javascriptvorgang zu schreiben.

Hierbei ist jedoch zu beachten, dass nicht der Vorgang auf den Wert eines Formularfeldes reagiert. Es wir dieses Javascript immer (wie im Auslöser definiert) ausgeführt, es wird nur innerhalb des JavaScript je nach Wert eine andere Aktion ausgeführt.

5.2. Zusammenfassung

Bei Vorgängen handelt es sich um vordefiniert Abläufe, die per Klick konfigurierbar sind, und zu einem bestimmten Zeitpunkt je nach Auslöser angestoßen werden.

Mit diesen Vorgängen ist es schnell und einfach und ohne weitere Kenntnisse einer Programmiersprache möglich, ein .pdf Dokument während der Laufzeit zu verändern.

Es können hier Werte geändert, Menüpunkte ausgeführt, oder auch Felder unsichtbar gemacht werden.

Ein prinzipielles Problem dieser einfachen Veränderung ist jedoch, dass nicht auf andere Felder reagiert werden kann.

Es ist nicht möglich nach einem Klick auf einen Button ein Feld unsichtbar zu machen wenn ein anderes Feld einen Wert größer 5 enthält, weil man nicht auf den Wert 5 zugreifen kann.

Es ist aber möglich mehrere Aktionen mit einen Auslöser zu kombinieren.

6. JavaScript

Dieses Kapitel beschreibt die Möglichkeiten mit der ab Acrobat Adobe 5.0 integrierten Schnittstelle Javascript Programme zu schreiben und diese auf bestimmte Auslöser und Werte reagieren zu lassen.

6.1. Einbindung eines JS

Es soll nun innerhalb von Adobe Acrobat ein Javascript erzeugt werden. Es gibt mehrer Möglichkeiten ein Javascript innerhalb des Dokuments zu erstellen.

6.1.1. Dokumenten Javascript

In diesem Fall wird ein „Dokumenten-JavaScript" verfasst. Alle so verfassten Javascripts können in anderen Javascript Methoden des Dokuments verwendet werden.

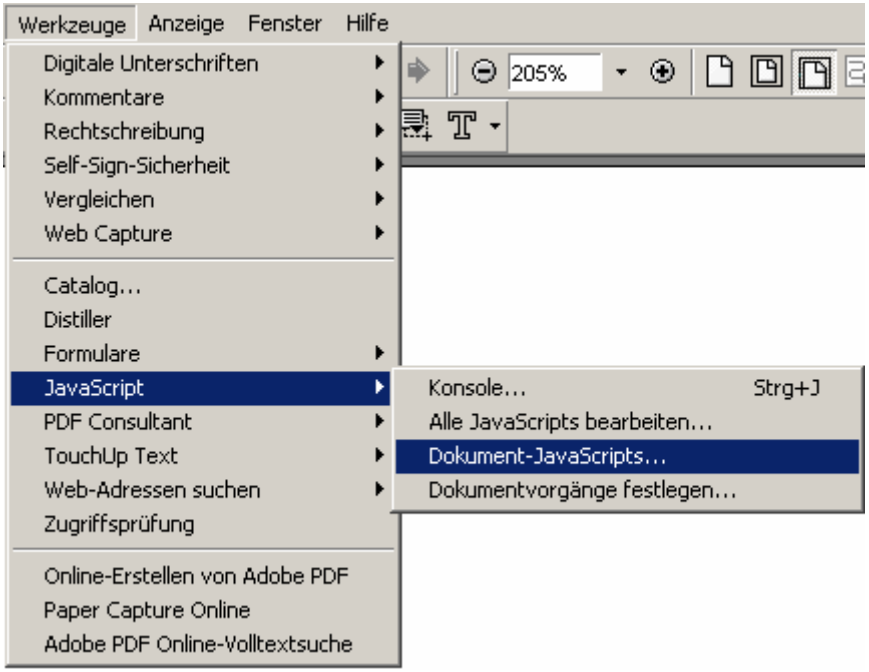


Abbildung 19: Javascriptfunktionen in Acrobat Adobe

Wird das „Dokument-JavaScript" welches sich unter „Werkzeuge" → „JavaScript" → „Dokument-JavaScript" befindet ausgewählt, so wird in dem geöffnetem Fenster ein Name für eine neue Methode angegeben. Durch den Button „Hinzufügen" wird ein Fenster geöffnet, in dem die neue Methode ausprogrammiert wird. Diese Methoden sind global. Was bedeutet, dass man auf sie aus jedem Javascript welches in das Dokument integriert wird, zugreifen kann.

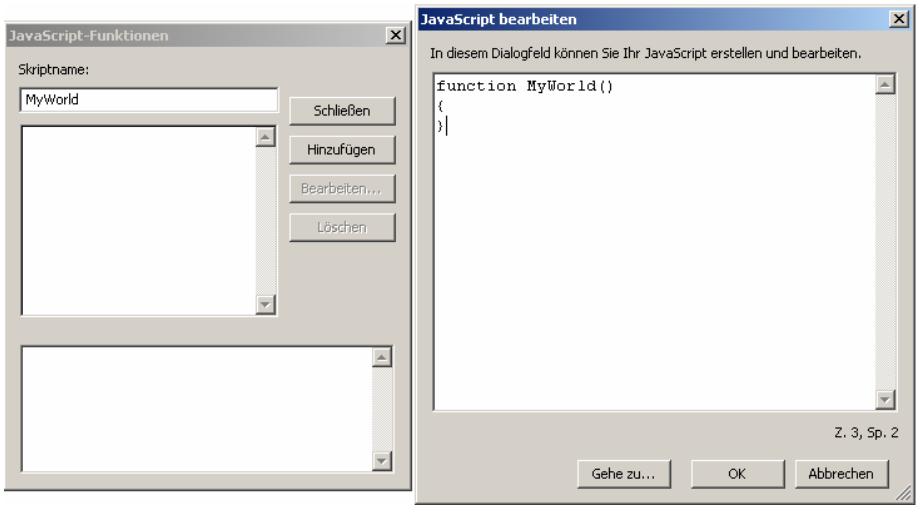


Abbildung 20: Dokument-Javascript

6.1.2. Javascript in Formularen

Wie schon in Abschnitt 5.1.1 können bei Formularen Vorgänge die auf bestimmt Auslöser reagieren hinzugefügt werden. Hier wird nun Javascript eingebunden.

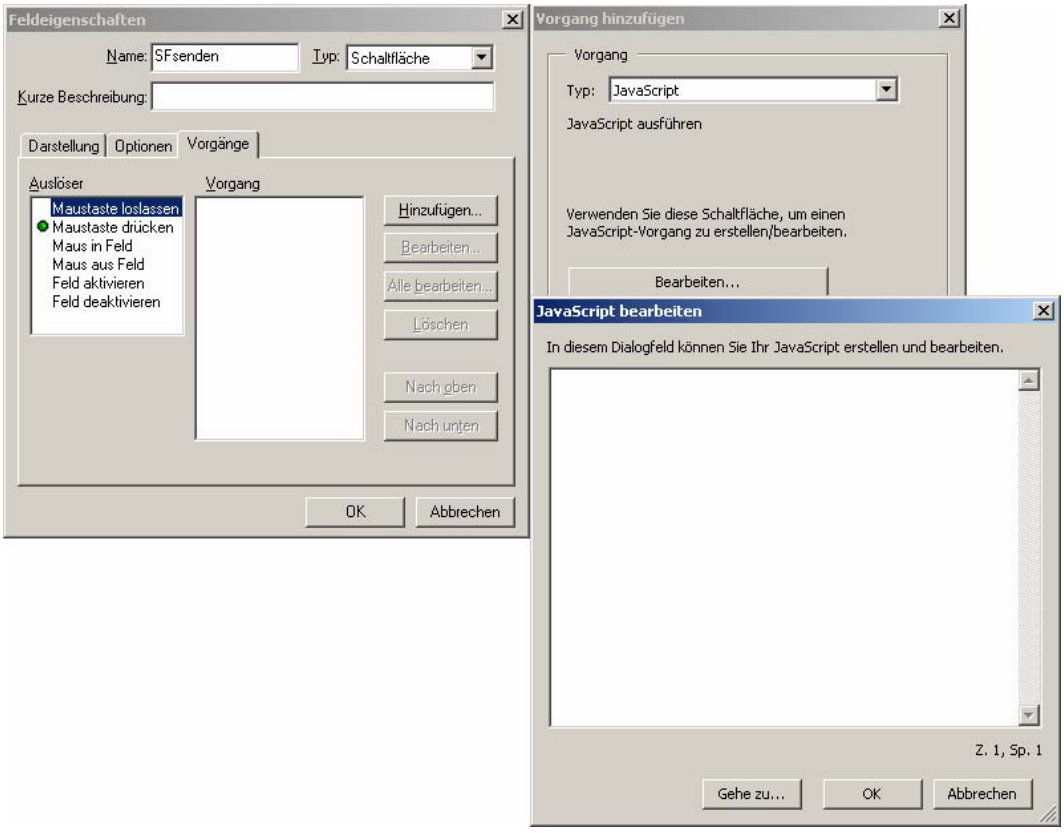


Abbildung 21: Javascript als Vorgang

6.1.3. Dokumentenvorgänge

Die dritte und letzte Stelle an der Javascript verwendet werden kann sind Dokumentenvorgänge. Dies sind Vorgänge die sich nicht auf ein bestimmtes Formularobjekt beziehen, sondern auf Vorgänge die das gesamte Dokument betreffen.

Eine nähere Beschreibung der Dokumentenvorgänge befindet sich in dem Abschnitt 6.1.3.

Die Vorgänge „Speichert Dokument" und „Hat Dokument gespeichert" sowie „Druckt Dokument" und „Hat Dokument gedruckt" werden in ihrer gemeinsamen Funktion in Abschnitt 7.2 beschrieben.

6.2. Speicherzähler

Hierzu wird eine Javascript entwickelt, welches in ein Formularfeld die Anzahl der getätigten Speichervorgänge schreibt. Dieses Feld ist zu Demonstrationszwecken in dem Dokument sichtbar, kann jedoch wie in Abschnitt 4.3.2.1 beschrieben auch auf „Unsichtbar" gesetzt werden. Mit Hilfe dieses Feldes können andere Javascriptvorgänge zu einem bestimmten Zeitpunkt ausgeführt werden.

Um diesen Speicherzähler zu erstellen wird einen neues Formularfeld eingefügt, dass den Namen „Speicherzähler" hat, und vom Typ „Text" ist. Der Standardwert dieses Feldes wird auf „0"(Null) gesetzt.

Nun werden die Dokumentenvorgänge[3] geöffnet und ein neues Javascript zu dem Vorgang „Speichert Dokument" hinzugefügt.

```
Var fSave = this.getField(„Speicherzähler“);
fSave.value = fSave.value + 1;
```

Code 1: Speicherzähler

Mit dem Code wird eine neue Variable „fSave" erstellt und mit dem Befehl „this.getField" mit dem Feld „Speicherzähler" verbunden. Weiters wird der Variablen „fSave" der „value" (Wert) der sich in diesem befindet zugewiesen und um 1 erhöht.

6.2.1. Verwendung des Speicherzählers

Hierzu werden wir das Beispiel aus Abschnitt 5.1.2.1 folgendermaßen umbauen:

Wir entfernen den Vorgang „Formular zurücksetzen". Stattdessen wird ein Javascriptvorgang hinzugefügt der den Text der sich neben dem Kontrollkästchen befindet ändert wenn das Dokument öfter als zwei Mal gespeichert wird.

1. Als erstes muss das Textfeld in welchem sich der Text befindet durch ein Formularfeld mit dem Namen „Transportversicherung" ersetzt werden.

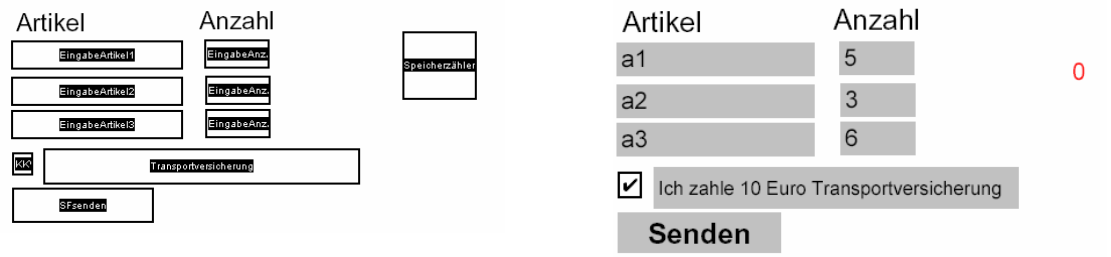


Abbildung 22: Formularfelder Beispiel

2. Der „Senden" Schaltfläche wir ein neuer Javascript Vorgang hinzugefügt.

[3] Pfad: Werkzeug → JavaScript → Dokumentenvorgänge festlegen…

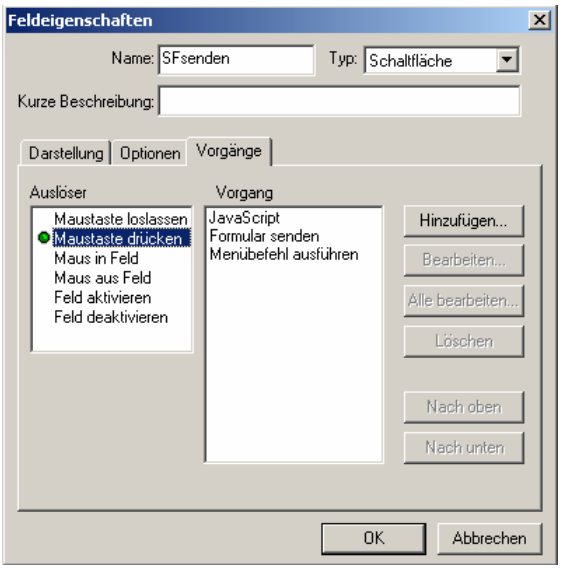


Abbildung 23: Neues Beispiel mit Javascript

3. Nun muss der Javascriptvorgang folgendermaßen implementiert werden

```
var fSave = this.getField(„Speicherzähler“)

if (fSave.value >= 2){
var fTransport = this.getField(„Transportversicherung“)
fTransport.value = „Ich zahle 100 Euro Transportversicherung“
}
```

Code 2: Speicherzähler Beispiel

Es wird der Variablen „fSave“ der Wert des Formularfelds „Speicherzäher“ zugewiesen. Danach wird in der „if“ Anweisung überprüft ob dieser größer oder gleich der Zahl zwei ist. Trifft dies zu, wird der Variablen „fTransport“ das Feld Transportversicherung zugewiesen. Danach wird der Variablen „fTransport“ der neue Text für das Feld Transportversicherung zugewiesen.

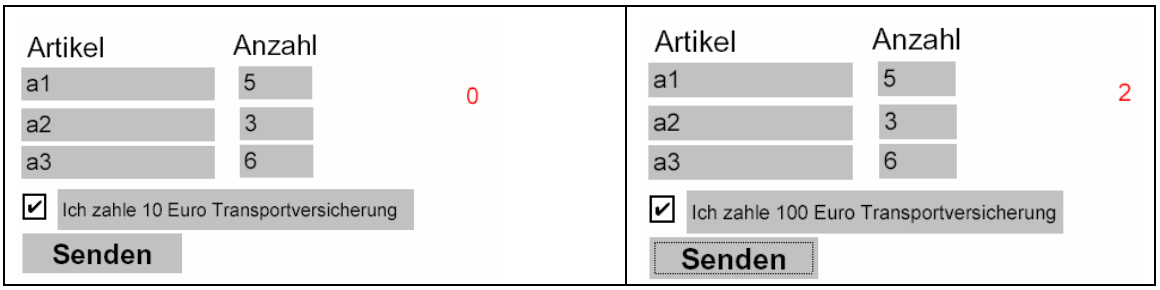


Abbildung 24: Endergebnis Beispiel

6.3. Erstellen von Datumsfeldern

Um die Veränderung nicht nur von der Speicheranzahl abhängig zu machen, kann auch das Datum überprüft werden. Dies geschieht am leichtesten über 3 Formularfelder des Typs „Text" wobei in eines das Jahr, in eines das Monat und in eines der Tag geschrieben wird.

Abbildung 25: Datumsfelder

In einem ersten einfachen Beispiel werden diese Datumsfelder durch einen Javascript Vorgang von einer Schaltfläche ausgelöst.

1. Erstellung drei neuer Felder mit den Namen „HeuteJ", „HeuteM" und „HeuteT"
2. Nun wird ein Button mit dem Namen „Datum" hinzugefügt.
3. Diese Schaltfäche bekommt einen Javascript Vorgang:

```
var fHeuteJ = this.getField(„HeuteJ“);
fHeuteJ.value = util.printd(„yyyy“, new Date());

var fHeuteM = this.getField(“HeuteM”);
fHeuteM.value = util.printd(“mm”, new Date());

var fHeuteT = this.getField(„HeuteT“);
fHeuteT.value = util.printd(“dd”, new Date());
```

Code 3: Datumsfelder

Hier wird jedem Feld das aktuelle Datum welches mit "new Date()" generiert wird zugewiesen. Durch den Aufruf mit „util" ist es möglich auch ein Format zu übergeben. Durch dieses Format wird immer nur Tag, Monat oder Jahr in das Feld geschrieben.

Durch die Trennung dieser Zeitfelder ist es im weiteren Verlauf einfacher auf ein bestimmtes Datum oder auch nur auf eine Monat zu überprüfen, als das gesamte Datum zu speichern und es erst im Nachhinein in seine Bestandteile zu zerlegen.

6.3.1. Verwendung von Datumsfeldern

Wie in dem obigen Beispiel schon beschrieben ist es jetzt möglich auch auf das Datum zu überprüfen. Dadurch kann ein Dokument zu einem beliebigen Zeitpunkt verändert werden.

```
var fHeuteJ = this.getField(„HeuteJ“);
fHeuteJ.value = util.printd(„yyyy“, new Date());

var fHeuteM = this.getField(“HeuteM”);
fHeuteM.value = util.printd(“mm”, new Date());

var fHeuteT = this.getField(„HeuteT“);
fHeuteT.value = util.printd(“dd”, new Date());
```

```
var fSave = this.getField(„Speicherzähler“)
if (fSave.value >= 2){
    if (fHeuteJ.value == 2006){
        var fTransport = this.getField(„Transportversicherung“)
        fTransport.value = „Ich zahle 100 Euro Transportversicherung“
    }
}
```

Code 4: Datumsfelder Beispiel

In diesem Beispiel wurde das Javascript welches auf der „Senden“ Schaltfläche liegt um die Option erweitert, so dass der neue Text nur in diesem Jahr verändert wird. Somit entsteht eine Kombination der Datumsfelder mit dem Speicherzähler.

Es wird die Variable „fSave“ überprüft ob sie größer oder gleich 2 ist. Wenn dies der Fall ist, wird überprüft, ob das aktuelle Jahr 2006 ist. Erst wenn beide Kriterien erfüllt sind, kommt es zu einer Änderung im Text.

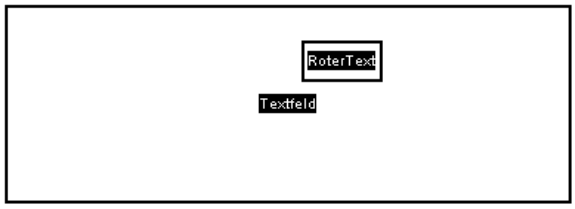
6.4. Ein-Ausblenden von Formularfeldern

Dies ist im Speziellen bei Formularfeldern des Typs „Text“ von Vorteil, da es sehr schwierig ist in dem Javascript den Text zu formatieren. Daher bietet es sich an Formularfelder zu erstellen und den Inhalt zu formatieren. Diese Felder müssen dann per Javascript nur noch ein oder ausgeblendet werden.

Weiters ist es nicht möglich in einem Formularfeld des Typs „Text“ verschiedene Farben oder Schriftarten zu verwenden. Soll dies realisiert werden, müssen verschiedene Textfelder übereinander gelegt und nacheinander eingeblendet werden.

6.4.1. Beispiel zum Einblenden von Formularfeldern

Im folgenden Beispiel werden zwei Formularfelder mit dem Namen „Textfeld“ und „RoterText“ erstellt.



Hierbei handelt es sich um eine mehzeiliges Textfeld, dessen Text auch rote Formatierung enthält.

Abbildung 26: Übereinander liegende Formularfelder

Das eine Formularfeld „Textfeld“ wird normal beschriftet.

Das zweite Formularfeld hat dieselbe Schriftart und Schriftgröße, jedoch eine andere Schriftfarbe und hat die Hintergrundfarbe „weiß“.

Nun muss das Formularfeld „RoterText“ genau in die den Bereich des Formularfeldes „Textfeld“ eingefügt werden.

Jetzt wird eine Schaltfläche mit dem Namen „Umschalten“ erstellt, welche einen Javascriptvorgang auf den Auslöser „Maustaste drücken“ enthält.

```
Var text = this.getField(“RoterText”)
if (text.hidden == false){
    text.hidden = true
}
else{
    text.hidden = false
}
```

Code 5: Ein-/ Ausblenden von Formularfeldern

Durch die Einführung dieses Javascriptcodes ist es erdenklich für den Bearbeiter den Text „rote“ in der Farbe rot darzustellen wenn dieser momentan in schwarz erscheint und auch umgekehrt.

Mit einem ähnlichen Javascripts wäre es auch möglich ganze Textpassagen bei einem Auslöser zu verändern ohne, dass der Bearbeiter es sieht.

6.5. Zusammenfassung

Um die Funktionen der vordefinierten Vorgänge zu erweitern, ist es unweigerlich notwendig einen Javascriptvorgang einzubauen und in diesem die gewünschte Funktionalität zu implementieren.

Der Javascriptvorgang an sich reagiert auf dieselben Auslöser wie die anderen Vorgänge auch, aber es sind hier noch zusätzliche Abfragen einbaubar, was mit anderen Vorgängen nicht möglich ist.

Nur so ist es denkbar auf bestimmte Werte in anderen Formularfeldern zu reagieren, oder verschiedene Aktionen aufgrund der Werte in anderen Feldern auszuführen.

Auch die Verwendung von „Dokumenten Javascripts“ die man als globale Funktionen bezeichnen könnte kann die Arbeit bei wiederkehrenden Abläufen wesentlich vereinfachen.

7. Vordefinierte Vorgänge

Dieses Kapitel behandelt Vorgänge die sich entweder auf eine Seite im Dokument oder auf Aktionen mit dem Dokument beziehen. Diese Vorgänge werden von Acrobat Adobe zur Verfügung gestellt und müssen mit entsprechenden Aktionen oder einem Javascript versehen werden

7.1. Seitenvorgänge

Seitenvorgänge sind Vorgänge die beim Verlassen oder Laden einer bestimmten Seite ausgeführt werden. Seitenvorgänge befinden sich unter „Dokument" → „Seitenvorgang festlegen…", oder in Acrobat Adobe 7.0 in den Eigenschaften einer Seite.

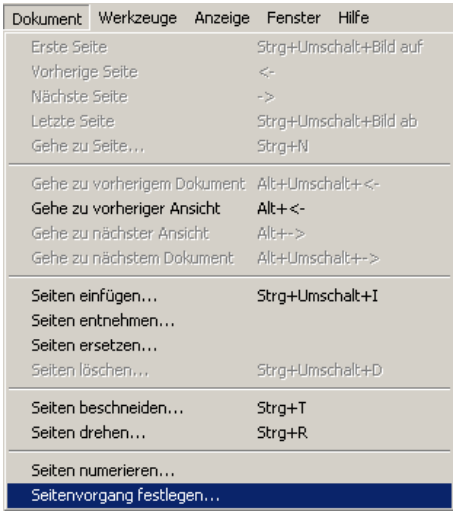


Abbildung 27: Seitenvorgang festlegen…

7.1.1. Öffnen Script

Es soll eine Aktion beim Öffnen des Dokuments ausgeführt werden. Dies ist jedoch nur über einen Umweg machbar, da es keinen Dokumentenvorgang zum Öffnen eines Dokumentes gibt. Aus diesem Grund wird ein Seitenvorgang auf die erste Seite des Dokuments gelegt.

Dieser Vorgang wird nun beim Laden der ersten Seite ausgeführt. Da das Dokument auch mit der ersten Seite geöffnet wird, kann in diesem Fall der Seitenvorgang als Dokumentenvorgang für das Laden des gesamten Dokuments verwendet werden.

Weiters kann dieses Script auch bestimmt Vorgänge ansteuern wenn eine Seite geladen wird. Hierbei ist jedoch zu beachten, dass es in Acrobat Adobe mehrere Optionen gibt wie man von einer Seite zur nächsten gelang. Mittels  ist es dem Benutzer möglich dies zu steuern. Im Folgenden werden diese 3 Alternativen von links beginnend beschrieben

- Einzelne Seite: Hierbei wird nach dem Ende der vorherigen Seite zu der nächsten gesprungen. Bei Auswahl dieser Option ist es möglich mittels eines Seitenvorganges auf der gleichen Seiten eine Veränderung durchzuführen, da das Script geladen wird bevor der Benutzer die Seite sieht.
- Fortlaufend: Bei Auswahl dieser Option gibt es keinen Sprung zur nächsten Seite, sondern diese wird fortlaufen in das Bild gebracht. Das Problem dieser Option ist, dass das Script nicht ausgeführt wird wenn die Seite am

Bildschirm erscheint, sondern wenn sie vollständig am Bildschirm geladen ist. Dadurch sieht der Benutzer einen Teil der Seite bevor das Script ausgeführt ist. Hierbei ist es nur möglich eine Seite vorher zu editieren ohne dass der Benutzer es sieht.

- Fortlaufend – Doppelseiten: Wenn diese Ansicht ausgewählt ist, kommt es zu keinem Aufruf der Seitenvorgänge.

Aufgrund dieser Einschränkung kann ein Dokument mittels Seitenvorgängen nicht verändert werden, da nicht sichergestellt werden kann zu welchem Zeitpunkt und ob der Dokumentenvorgang ausgeführt wird.

7.1.1.1. Verwendung des Öffnen Scripts

Obwohl wie im vorigen Kapitel beschrieben das Festlegen eines Seitenvorganges in einem Dokument nicht von Vorteil ist um dieses zu verändern, bietet es sich dennoch an, auf die erste Seite einen Seitenvorgang zu legen um das Öffnen eines Dokuments (bei welchem die erste Seite angezeigt wird und somit auch der Vorgang der ersten Seite ausgeführt wird) abzufangen und darauf zu reagieren.

Zu Beispielzwecken ist es möglich auf die erste Seite des Dokuments einen Seitenvorgang mit folgendem Javascript zu implementieren.[27]

```
App.alert(„öffnen“)
```

Code 6: Öffnen Script Beispiel

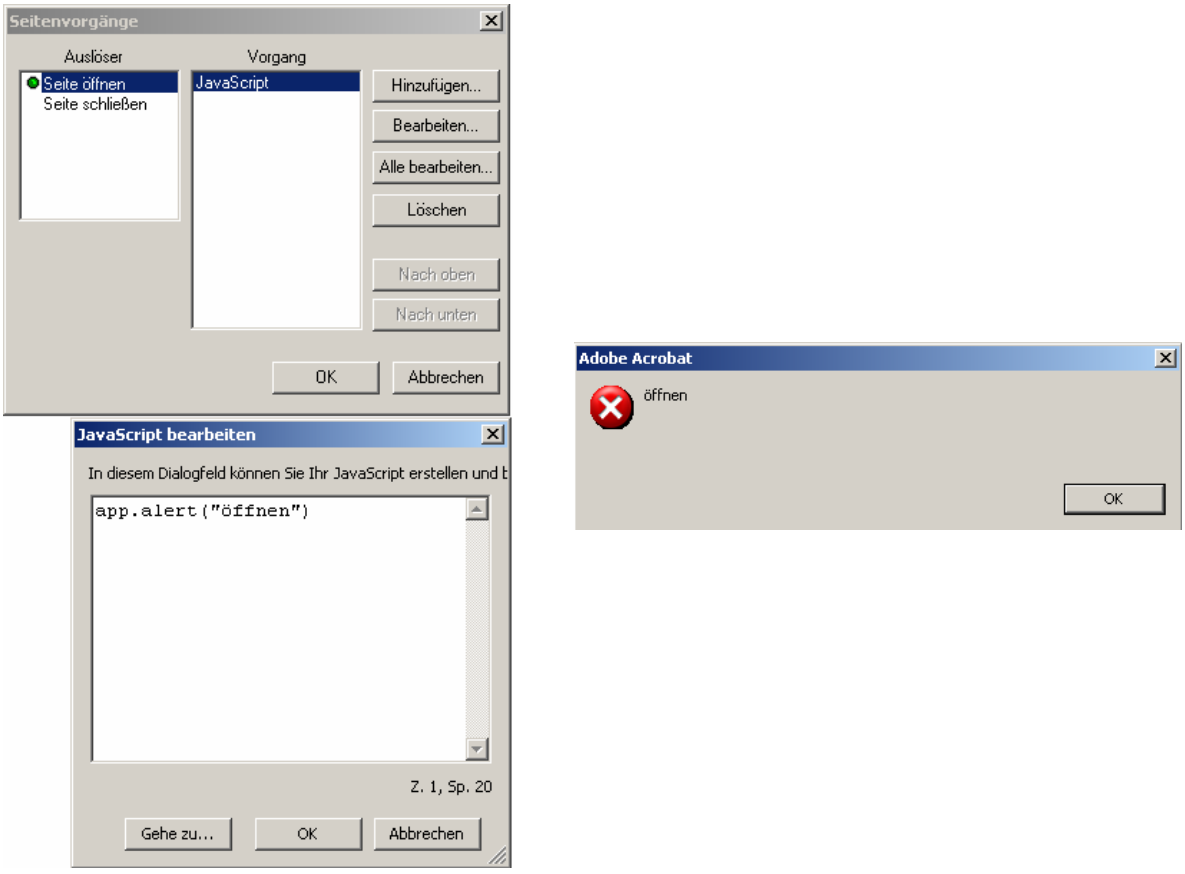


Abbildung 28: Öffnen Script

Durch die Verwendung des Öffnen Scripts ist es denkbar auf bestimmte Faktoren zu reagieren oder dieses zu setzten.

Durch die Verwendung eines Datumsfeldes in Zusammenhang mit dem Speicherzähler wäre es erreichbar das erste Öffnen einen Dokuments zu speichern, oder Veränderungen an einem Dokument von dem momentanen Datum oder Anzahl der Öffnungen abhängig zu machen. Mit dem Öffnen Script ist es auch möglich in einem Formular einen Vorgabewert, welcher sich aufgrund anderer Werte und Formularfelder ergibt zu setzten.

7.1.2. Schließen Script

Dabei handelt es sich um das Gegenstück zu dem Seitenvorgang der beim Laden einer Seite ausgeführt wird. Das hier hinterlegte Script wird beim Verlassen einer Seite durchgeführt. Es gelten dieselben Regeln und Unsicherheitsfaktoren wie bei dem Öffnen Script.

7.2. Dokumentenvorgänge

Wie auch Seitenvorgänge werden diese Vorgänge von Acrobat Adobe zur Verfügung gestellt und müssen nun mit einer Action oder einem Javascript versehen werden. Die Dokumentenvorgänge finden sich unter „Werkzeuge" → „JavaScript" → „Dokumentenvorgänge festlegen…"

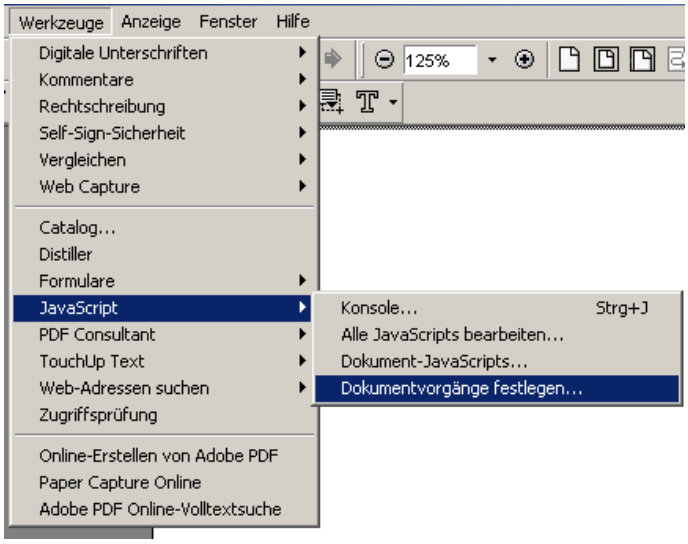


Abbildung 29: Ort der Dokumentenvorgänge

7.2.1. Speichert Dokument

Durch diesen Vorgang ist es erreichbar eine Aktion oder ein Javascript auszuführen bevor ein Dokument gespeichert wird. Eine mittels dieses Vorganges durchgeführte Änderung befindet sich sowohl in dem abgespeicherten Dokument als auch in dem Dokument welches der Benutzer sieht. Somit ist es zwar möglich eine Veränderung beim Speichern herbeizuführen, jedoch ist diese Änderung auch für den Benutzer sichtbar.

Dieser Vorgang gilt global für alle Speicherungen des Dokuments. Es wird also sowohl ausgeführt wenn der Benutzer das Dokument über die Schaltflächen in der Kopfleiste als auch über „Datei" →"Speicher" und „Datei" → „Speichern unter…" und auch beim Einfügen einer „Speichern" Schaltfläche in das Dokument.

7.2.2. Hat Dokument gespeichert

Hierbei handelt es sich um einen Vorgang, welcher nach dem Speichern eines Dokuments durchgeführt wird. Dieser Vorgang verändert lediglich was der Benutzer auf dem Bildschirm sieht, jedoch nicht das abgespeicherte Dokument.

7.2.3. Druckt Dokument

Dieser Vorgang hat dieselben Eigenschaften wie der Vorgang 7.2.1, jedoch wird er nicht auf das Speichern des Dokuments ausgeführt sondern auf das Drucken. Auch hier ist es möglich den Text für Felder zu verändern und dieses zu Drucken und dem Benutzer nach dem Druck anzuzeigen.

7.2.4. Hat Dokument gedruckt

Auch hier handelt es sich um einen ähnlichen Vorgang wie 7.2.2 wobei sich der Vorgang nicht auf das Speichern eines Dokuments bezieht, sondern auf das Drucken.

7.2.5. Kombinierte Dokumentenvorgänge

Um ein Dokument zu verändern und es dem Benutzer am Bildschirm nicht zu zeigen ist es von Vorteil die oben genannten Vorgänge zu kombinieren. Es wird jeweils ein Vorgang welcher vor dem Speichern und der dazugehörige Vorgang nach dem Speichern kombiniert. Dadurch ist es möglich ein Dokument vor dem Speichern oder Drucken zu verändern, und es nach dem Drucken oder Speichern dem Benutzer wieder in Original zu zeigen. Damit hat der Benutzer keine Kontrolle darüber was gedruckt oder gespeichert wird.

Dazu werden zwei Formularfelder benötigt. Das Feld „Summe" und das Feld „SummeAlt" wobei es sich bei dem Feld „SummeAlt" um ein Feld handelt, welches der Benutzer nicht sieht. Diese beiden Formularfelder sind von Typ „Text" und als Zahlenfelder formatiert.

Abbildung 30: Dokumentenvorgang kombiniert Formularfelder

Nun soll der Wert den der Benutzer in das Feld „Summe" einträgt in einem ersten Schritt in das Feld „SummeAlt" kopiert werden, um den Wert in dem Feld „Summe" mit 1,1 zu multiplizieren. Dann wird das Dokument mit dem neuen Wert gespeichert.

In einem zweiten Schritt wird der Wert aus dem Feld „SummeAlt" wieder in das Feld „Summe" geschrieben. Dadurch wird ein Wert gespeichert den der Benutzer nie sieht.

Das gleiche Vorgehen kann auch für den Druck verwendet werden.

Hierzu wird nun der folgende Javascript in den Dokumentenvorgang „Speichert Dokument" eingefügt.

```
Var fSumAlt = this.getField(“SummeAlt”);
var fSum = this.getField(“Summe”);


fSumAlt.value = fSum.value;
fSum.value = fSum.value *1.1;
```

Code 7: Kombinierte Formularfelder 1

Hier wird der Variablen „FsumAlt" das Feld SummeAlt zugewiesen und der Variablen „fSum" das Feld Summe. Danach wird der Variablen „fSumAlt" der Wert der Variablen „fSum" zugewiesen. Abschließend wird die Variable „fSum" mit dem Wert 1,1 multipliziert. Somit steht in dem Feld Summe, welches für einen Benutzer sichtbar ist der neue erhöhte Wert, und in dem unsichtbaren Feld SummeAlt wurde der alte Wert gespeichert.

Weiters wird bei dem Dokumentenvorgang „Hat Dokument gespeichert" das folgende Javascript eingetragen.

```
Var fSumAlt = this.getField("SummeAlt");
var fSum = this.getField("Summe");


fSum.value = fSum.valueAlt;
```

Code 8: Kombinierte Formularfelder 2

Mit Hilfe dieses Javascript wird der ursprüngliche Wert wieder in das Feld „Summe" geschrieben.

Durch dieses Vorgehen wird ein Dokument gespeichert oder gedruckt was nicht dem entspricht was der Benutzer sieht.

Es ist auch möglich in das Feld „Summe" nicht nur einen Wert zu schreiben sondern diesen Wert aus einer Reihe andere Felder zu errechnen. Somit kann der Bestellwert verschiedener Produkte errechnet und verändert werden.

7.3. Zusammenfassung

Es ist möglich in einem .pdf Dokument nicht nur auf direkte Aktionen eines Benutzers, sondern auch auf indirekte Aktionen zu reagieren. Somit kann nicht nur auf einen Klick auf einen Button reagiert werden sondern auch auf Umstände wie das Erreichen oder das Verlassen einer Seite oder auch auf das Speichern, Drucken und Schließen eines Dokuments.

Zu jedem dieser Vorgänge kann ein eigenes Javascript implementiert werden, welches unter Umständen auch auf zuvor gespeicherte Werte wie „wie oft wurde das Dokument schon gespeichert" zugreift.

Im Speziellen bei Seitenvorgängen muss man aufpassen, da diese je nach Art wie der Benutzer von Seite zu Seite gelangt erst sehr spät (große Teile der Seite schon sichtbar) oder auch gar nicht ausgeführt werden.

Die Dokumentenvorgänge, welche sich auf das Schließen, Drucken und Speichern einer Seite beziehen werden zu einem sehr genau definierten Zeitpunkt ausgeführt. Hier fehlt lediglich ein Vorgang zum Öffnen des Dokuments, was aber mit einem Seitenvorgang auf das Öffnen der ersten Seiten (= Öffnen des Dokuments) realisiert werden kann.

Bei den Vorgängen für das Speichern und das Drucken wird auch noch zwischen dem Zeitpunkt davor und dem Zeitpunkt danach unterschieden.

8. Änderungen im .pdf Code

Bei dieser Variante wird nicht die in Acrobat Adobe integrierte Javascript Konsole verwendet, sondern es wird direkt in dem Code der .pdf Datei ein Javascript eingetragen.

Wenn ein Javascript in Acrobat Adobe eingegeben und dann gespeichert wird kommt es zu einer Formatierung mancher Zeichen. Dies muss bei Erstellung eines Javascript im Code manuell gemacht werde. Als Alternative dazu ist es möglich in einem Javascript ein `stream` Objekt einzuführen um die Sonderzeichen nicht markieren zu müssen.

Bei Verwendung einer `streams` wird der Anfang des `streams` mit dem Wort `stream` und das Ende mit dem Wort `endstream` gekennzeichnet.

8.1. Objekte

Um ein Javascript in den Code eines .pdf Dokuments einbauen zu können ohne das Dokument zu öffnen ist es notwendig zu verstehen wie ein .pdf Dokument aufgebaut ist.

Es gibt die einfache Zuweisung von Werten. Dies geschieht hauptsächlich in den ersten Zeilen eines .pdf Dokuments. Finden sich diese Zuweisungen nicht am Anfang eines .pdf Dokuments ist dies eine Indiz dafür, dass das Dokument verändert wurde (siehe 8.3 Historisierung).

Die grundlegende Vorgehensweise ist jedoch die Zuweisung von Objekten. Hierbei wird einem Schlüssel ein bestimmtes Objekt zugeordnet.

```
/Info 4 0 R
/Root 7 0 R
Prev 14477
/ID[<64df757458631
>>
stratxref
0
%%EOF

7 0 obj
<<
/Type /Catalog
/Pages 3 0 R
/Metadata 5 0 R
/PageLabels 2 0 R
```

Code 9: Objektzuordnung

In Code 9 ist ersichtlich, dass dem Schlüssel „/Root" welcher durch das vorangegangene Zeichen „/" geschützt ist das Objekt „7 0" zugeordnet ist. In dem Objekt „7 0" werden weitere Objekte aufgerufen.

```
3 0 obj
<<
/Type /Pages
/Kids [8 0 R]
/Count 1
>>
endobj
```

Code 10: Objekt „Pages"

Wie in Code 10 ersichtlich wird dem Schlüssel „/Pages" das Objekt „3 0" zugeordnet. In dem Objekt „3 0" wird der Type und auch die Anzahl der Seiten festgelegt. Weiters gibt es hier einen Schlüssel „/Kids" welcher wieder auf ein Objekt verweist.

```
8 0 obj
<</Type /Page
/Parent 3 0 R
/Resources 9 0 R
/Contents 12 0 R
/MediaBox [0 0 595 842]
/CropBox [0 0 595 842]
/Rotate 0
```

Code 11: Seitenobjekt

Das Objekt „8 0" ist nun das Objekt welches die Attribute für die erste Seite in dem .pdf Dokument festlegt. Nachfolgend wird in diesem Objekt ein Seitenvorgang für die erste Seite des .pdf Dokuments eingetragen.

Wie aus Code 10 unter dem Schlüssel „/Count" ersichtlich, besteht dieses Dokument aus nur einer Seite. Wenn es sich um ein Dokument mit mehr als einer Seite handelt, so werden alle Seiten bei dem Schlüssel „/Kids" eingetragen. Wie in Code 12 ersichtlich, existieren hier drei Seiten, welche in den Objekten „16 0", „1 0", und „5 0" näher beschreiben werden. Die Reihenfolge der Initialisierung entspricht auch der Reihenfolge der Seiten. Somit wird mit Objekt „16 0" die Seite 1, mit Objekt „1 0 " Seite 2 und mit Objekt „5 0" Seite 3 beschrieben.

```
/Type /Pages
/Kids [16 0 R 1 0 R 5 0 R]
/Count 3
```

Code 12: Objekt mit drei „Pages"

8.1.1. Cross Reference Table

Die Cross Reference Table steht am Anfang jedes .pdf Dokumentes und ist ein Inhaltsverzeichnis innerhalb des .pdf Codes für bestimmte systemeigene Objekte, welche durch das Objektmodell in dem .pdf Code nicht erreicht sind oder oft verwendet werden und nicht nur durch das Lesen des gesamten Objektbaumes erreicht werden sollen. Somit ist die Reference Table nicht nur eine System-Tabelle sonder auch eine Beschleunigung der Performance.

```
Xref
14 8
0000000016 00000 n
0000000805 00000 n
0000001067 00000 n
0000001278 00000 n
0000001469 00000 n
0000001806 00000 n
0000002048 00000 n
0000000640 00000 n
```

Code 13: Referece Table

Diese Tabelle wird in einer .pdf Datei immer mit dem Tag *xref* angezeigt. In der ersten Zeile befinden sich immer zwei Zahlen die durch ein Leerzeichen getrennt sind.

Die erste Zahl gibt an auf welches Objekt die erste Zeile in der Cross Reference Table verweist. Die zweite Zahl gibt die Anzahl der Einträge in der Tabelle an.

Nach dieser Zeile, mit der Angabe des ersten Objektes und des Beginns folgen die Einträge der Tabelle an sich.

Diese Kombination kann sich beliebig oft wiederholen, wie in Code 14 ersichtlich ist.

```
xref
6 1
0000007967 00000 n
15 2
0000008243 00000 n
0000008522 00000 n
23 10
0000008747 00000 n
0000008889 00000 n
0000008965 00000 n
0000009054 00000 n
0000010245 00000 n
0000010422 00000 n
0000010446 00000 n
0000010540 00000 n
0000010753 00000 n
0000000000 65535 f
```

Code 14: Erweiterte Cross Reference Table

Weiters wird diese Tabelle auch bei jedem Speichern neu geschrieben um auf die neuen Objekte zu verweisen.

Die Einträge stehen pro Zeile für genaue in Objekt und sind genau 20 Bytes lang. Es unterscheiden sich 2 Arten von Einträgen, welche durch das letzte (zwanzigste) Byte bestimmt werden. Dieses Byte kann den Eintrag *n* oder *f* haben.

Der Eintrag *n* steht dafür, dass dieses Objekt „in-use" also gebraucht ist. Ist der Eintrag jedoch auf *f* so steht dies für „free", das Objekt ist im Moment nicht gebraucht.

Handelt es sich um einen Eintrag mit dem Flag n, so ist der Eintrag folgendermaßen zu lesen:

aaaaaaaaaa bbbbb

aaaaaaaaaa gibt in Bytes an, an welcher Stelle das gewünscht Objekt beginnt.

bbbbb ist eine einfache Nummer die nicht befüllt werden muss.

Ist der Eintrag jedoch mit dem Flag *f* versehen ist die wie folgt zu interpretieren:

aaaaaaaaaa bbbbb

aaaaaaaaaa ist der Platz des nächsten freien Objektes

bbbbb ist wie bei dem Flag n auch eine Nummer. Ist ein Objekt jedoch gelöscht worden und noch in dem Code vorhanden, so wir über diese Nummer angegeben, welche Objektnummer das neue Objekt desselben Typs bekommen soll. Wie in Code 15 ersichtlich soll für das Objekt 3 als nächstes die Nummer sieben (7) vergeben werden. Gibt es keine vordefinierte Nummer so wir der Eintrag mit der Zahl 65553 befüllt.

```
Xref
0 6
0000000003 65535 f
0000000017 00000 n
0000000081 00000 n
0000000000 00007 f
0000000331 00000 n
0000000409 00000 n
```

Code 15: Cross Reference Table mit gelöschtem Eintrag

8.2. Vordefinierte Vorgänge

Abgesehen von Javascript gibt es eine Reihe von vordefinierten Vorgängen die Adobe Acrobat zu Verfügung stellt. Auch diese Vorgänge können in dem Code eines .pdf Dokuments gesehen werden. Diese Vorgänge liegen in einem eigenen Objekt, welches von einem Formular referenziert wird. Folgende Schlüssel definieren diese Vorgänge welche in einem Objekt unter Schlüssel „/S" zu finden sind (z.B.: /S /JS):

- /Thread: Definiert den Vorgang „Artikel lesen"
- /Sound: Definiert den Vorgang „Audio"
- /Launch: Definiert den Vorgang „Datei öffnen"
- /Hide: Definiert den Vorgang „Feld ein-/ausblenden"
- /SubmitForm: Definiert den Vorgang „Formular senden"
- /ResetForm: Definiert den Vorgang „Formular zurücksetzen"
- /ImportData: Definiert den Vorgang „Formulardaten importieren"
- /JavaScript: Definiert den Vorgang „Javascript"
- /Name: Definiert den Vorgang „Menübefehl ausführen"
- /Movie: Definiert den Vorgang „Movie"
- /URI Definiert den Vorgang „WWW-Verknüpfung"

8.3. Historisierung

Nachdem nun das Objektmodell in .pdf Dokumenten bekannt ist, ist es auch möglich aus dem Code eines .pdf Dokuments alle Änderungen zu sehen die durchgeführt wurden. Als Voraussetzung dafür gilt jedoch, dass das Dokument mit der „Speichern" Schaltfläche, oder unter „Datei" → „Speichern" gespeichert wird. Wenn das .pdf Dokument mit dem Befehl „Datei" → „Speicher unter…" gespeichert wird, wird das .pdf Dokument komplett überschrieben, und Änderungen sind nicht nachvollziehbar.

Die Historisierung ist folgendermaßen aufgebaut:

- Der Schlüssel „/Root" muss sich am Anfang (vor dem ersten codierten Teil) befinden. Ist dies nicht der Fall ist das ein eindeutiges Indiz dafür, dass das Dokument verändert und gespeichert wurde.
- In jedem .pdf Dokument gibt es einen Bereich mit Benutzerinformationen. Für diesen Bereich gilt dasselbe wir für den „/Root" Schlüssel. Dieser Teil muss sich vor dem ersten codierten Teil befinden.

```
/CreationDate (D:20070211120928Z)
/ModDate (20070217104846+01'00')
/Producer (Acrobat Distiller 5.0 \(Windows\))
/Author (Christoph)
/Creator (Pscript5.dll Version 5.2.2)
/Title (Microsoft Word – 1.doc)
```

Code 16: Historisierung

- Der aktuelle Teil eines .pdf Dokuments befindet sich immer in jenem Teil des Codes zwischen Benutzerinformationen und dem Ende oder dem nächsten Abschnitt der Benutzerinformationen im welchem sich der „/Root" Schlüssel befindet.
- In dem aktuellen Teil dieses .pdf Dokuments befinden sich auch immer die Änderungen die in diesem Schritt durchgeführt wurden. Jedoch werden die alten Objekte nicht gelöscht (mit Ausnahme des „/Root" Schlüssels). Dadurch kommt es zum Vorhandensein von Objekten mit derselben Nummer wobei immer nur ein Objekt das derzeit gültige ist.

Letzte Speicherung	Alte Speicherung
<<	<<
/CreationDate(D:20070211120928z)	/CreationDate(D:20070211120928z)
/ModDate(D:20070217124846+01'00')	/ModDate(D:20070217102500+01'00')
/Producer(Acrobat Distilled 5.0 \(Windows\))	/Producer(Acrobat Distilled 5.0 \(Windows\))
/Author (Christoph)	/Author (Christoph)
/Creator (Pscript.dll Version 5.2.2)	/Creator (Pscript.dll Version 5.2.2)
/Title (Microsoft Word – 1.doc)	/Title (Microsoft Word – 1.doc)
>>	>>
Endobj	Endobj
7 0 obj	/Type /Catalog
<<	/Pages 3 0 R
/Type /Catalog	/Metadata 25 0 R
/Pages 3 0 R	/PageLabels 2 0 R
/Metadata 31 0 R	/AA <</WC 18 1 R /WS 21 0 R /DS 22 0 R /WP 23 0 R /DP 24 0 R>>
/PageLabels 2 0 R	>>
/AA <</WC 18 1 R /WS 21 0 R /DS 22 0 R /WP 23 0 R /DP 24 0 R>>	Endobj
/Names 26 0 R	18 1 obj
Endobj	<<
18 1 obj	/S /JavaScript
<<	/JS (app.alert\("Speichert Dokument"\))
/S /JavaScript	>>
/JS (app.alert\("Schließt Dokument Teil2"\))	Endobj

>> Endobj 30 0 obj /Size 32 /Info 4 0 R /Root 7 0 R /Prev 19346 /ID[64df5754586319…	21 0 obj << /S /JavaScript /JS (app.alert\(“Schließt Dokument ”\)) >> Endobj 22 0 obj << /S /JavaScript /JS (app.alert\(“Hat Dokument gespeichert”\)) >> Endobj 23 0 obj << /S /JavaScript /JS (app.alert\(“Druckt Dokument ”\)) >> Endobj 24 0 obj << /S /JavaScript /JS (app.alert\(“Hat Dokument gedruckt”\)) >> Endobj

Tabelle 5: Historisierungsbeispiel

In Tabelle 5 handelt es sich um zwei Codebeispiele aus demselben Dokument, wobei sich der Teil „Alte Speicherung“ über dem Teil der „Letzten Speicherung“ befindet. Wie auch ersichtlich ist, gibt es das Objekt „18 0“ zwei mal in diesem Dokument. Um nun das Aktuelle zu finden, muss ausgehend vom „/Root“ Schlüssel nach oben gesucht werden. Für das Objekt „18 0“ wird das Objekt aus „Letzte Speicherung“ verwendet. Wird jedoch nach dem Objekt „21 0“ gesucht welches in letzte Speicherung nicht existiert, wird dieses Objekt aus „Alte Speicherung“ verwendet.

Weiters ist hier in „Alte Speicherung“ zu sehen, dass die Objekte „18 0“, „21 0“, „22 0“, „23 0“ und „24 0“ am 17.02.2007 um 10:25:00 von „Christoph“ hinzugefügt wurden.

Das Objekt „18 0“ wurde am 17.02.2007 um 10:48:26 von „Christoph“ verändert und erneut gespeichert.

- Um also die aktuellen Änderungen zu finden, muss der .pdf Code ausgehend vom „/Root“ Schlüssel von unten nach oben gelesen werden. Das jeweils erste passende Objekt das auf diese Weise gefunden wird ist das aktuelle.

Auch wenn das .pdf Dokument mit „Speichern unter…“ gespeichert wird ist aber ersichtlich wann dies von wem gemacht wurde. Die einzige Möglichkeit dies zu umgehen

ist, den „/Author“ und die Zeit nachträglich zu ändern oder die Änderungen nicht in Acrobat Adobe sondern in dem .pdf Code durchzuführen.

Hierbei ist jedoch zu sagen, dass im Hinblick darauf, dass ein Dokument verschlüsselt wird, folgendes Verhalten auftritt:

Es kommt zwar zu einer Historisierung, aber zu dem Zeitpunkt wo das Dokument verschlüsselt wird, wird auch die zuvor erstellt Historisierung noch einmal überarbeitet.

Es wird der Teil der zuvor die Historisierung war und Klartext, z.B. ein Javascript, nun auch noch einmal überschrieben, und nicht mehr in Klartext gespeichert.

Wird also ein Dokument gesichert, so ist auch die Historisierung der Teile die zuvor gemacht wurden und zu diesem Zeitpunkt noch in Klartext waren nicht mehr lesbar.

8.4. Seitenvorgänge in Objekten

Auch Seitenvorgänge werden in dem Objektmodell von .pdf Dokument dargestellt. Dies geschieht nicht in codierter Form sondern in Klartext. Um nun zu sehen ob sich ein Seitenvorgang auf der ersten Seite des Dokuments befindet muss das Objekt der ersten Seite gesucht werden. Wie aus Code 12 ersichtlich, handelt es sich bei dem Objekt der ersten Seite um Objekt „16 0“.

```
16 0 obj
<<
/Type /Page
/Parent 11 0 R
/Resources 17 0 R
/Contents 21 0 R
/Mediabox [0 0 595 842]
/CropBox [0 0 595 842]
/Rotate 0
/AA <</O 25 0 R>>
>>
endjob
```

Code 17: Seitenvorgang in „Page“ Objekt

Wichtig ist der Schlüssel „/AA“ um den Seitenvorgang zu finden. Hier wird zwischen doppelten Spitzklammern auf ein neues Objekt verwiesen. Vor diesem Objekt wird mittels eines Schlüssels der Typ des Seitenvorganges festgelegt:

- **/O**: Dieser Schlüssel kennzeichnet das folgende Objekt also Vorgang „Seite öffnen“.
- **/C**: Dieser Schlüssel kennzeichnet das folgende Objekt also Vorgang „Seite Schließen“.

Wie aus Code 17 ersichtlich wird das Objekt „25 0“ beim Öffnen der Seite angezeigt.

```
25 0 obj
<<
/S /JavaScript
/JS (app.aler\(„eins“\))
>>
endobj
```

Code 18: Javascript Objekt

Das in Code 18 dargestellt Objekt beinhaltet einen Meldung beim Öffnen der Seite welche wie in Abbildung 31 gezeigt aussieht.

Abbildung 31: Adobe Acrobat Alert

8.5. Dokumentenvorgänge in Objekten

Wie auch Seitenvorgänge können Dokumentenvorgänge in dem .pdf Code hinzugefügt werden. Um einen Dokumentenvorgang hinzuzufügen muss wieder das „/Root" Element gesucht werden. Danach das Objekt auf welches das „/Root" Objekt verweist. In diesem Objekt kann mittels des Schlüssels „/AA" ein Dokumentenvorgang eingefügt werden.

```
7 0 obj
/Type /Catalog
/Pages 3 0 R
/Metadata 31 0 R
/PageLabels 2 0 R
/AA << /WC 18 1 R /WS 21 0 R /DS 22 0 R /WP 23 0 R /DP 24 0 R
     >>
/Names 26 0 R
>>
Endobj
18 0 1 obj
<<
/S /JavaScript
/JS (app.alert\(“Schließt Dokument Teil2”\))
>>
endobj
<<
/Size 32
/Info 4 0 R
/Root 7 0 R
/Prev 19346
/ID [<64df7574586319…
>>
```

Code 19: Dokumentvorgänge in Objekten

Folgenden Schlüssel können für Dokumentenvorgänge verwendet werden:

- **/WC:** Dieser Schlüssel entspricht dem Dokumentenvorgang „Schließt Dokument"
- **/WS:** Dieser Schlüssel entspricht dem Dokumentenvorgang „Speichert Dokument"
- **/DS:** Dieser Schlüssel entspricht dem Dokumentenvorgang „Hat Dokument gespeichert"
- **/WP:** Dieser Schlüssel entspricht dem Dokumentenvorgang „Druckt Dokument"
- **/DP:** Dieser Schlüssel entspricht dem Dokumentenvorgang „Hat Dokument gespeichert"

8.6. Formularvorgänge in Objekten

Wie auch bei Dokumentenvorgängen ist es möglich für Formularfelder einen Javascriptvorgang in den Objekten festzulegen. Da es sich beim Erstellen eines Formularfeldes um einen komplexen Vorgang mit Referenzen handelt wird hier lediglich die Möglichkeit beschrieben in ein vorhandenes Formular einen Javascriptvorgang einzubinden.

8.6.1. Auslöser

Die Auslöser sind wie folgt zu verstehen und werden innerhalb von zwei doppelten Spitzklammern beschrieben:

- AUSNAHME: Bei der Einbindung einer Schaltfläche gibt es eine Ausnahme. Normal werden die Vorgänge wie bei Seiten- und Dokumentenvorgängen innerhalb des Schaltflächenobjekts mittels des Schlüssels „/AA" und zwischen zwei doppelten Spitzklammern eingetragen. Bei dem Vorgang „Maustaste loslassen" muss jedoch ein eigener Schlüssel „/A" angelegt werden.

```
18 0 obj
<<
/Type /Annot
/Subtype /Widget
/Rect [ 131.22041 724.43822 185.85493 744.92616 ]
/F4
/P 13 0 R
/DA (/HeBo 12 Tf 0 g)
/MK <</N 19 0 R /D 20 0 R >>
/FT /Btn
/AA <</D 33 0 R /E 29 0 R /X 26 0 R /Fo 27 0 R /B1 28 0
R>>
/A 31 0 R
>>
```

Code 20: Formular Schaltfläche

Weitere Vorgänge werden innerhalb der Schlüssels „/AA" mit folgenden Schlüsseln versehen:

- /D: Dieser Schlüssel beschreibt in einer Schaltfläche den Vorgang „Maustaste drücken".
- /E: Dieser Schlüssel beschreibt in einer Schaltfläche den Vorgang „Maus in Feld".
- /X: Dieser Schlüssel beschreibt in einer Schaltfläche den Vorgang „Maus aus Feld".
- /Fo: Dieser Schlüssel beschreibt in einer Schaltfläche den Vorgang „Feld aktivieren".
- /Bl: Dieser Schlüssel beschreibt in einer Schaltfläche den Vorgang „Feld deaktivieren".

8.7. Zusammenfassung

Durch den Aufbau des .pdf Codes in Objekten ist es einfach möglich, vorausgesetzt man kennt das Objektmodell, weitere Javascript zu dem Code hinzuzufügen ohne dass ein Programm zum Öffnen des Dokuments zu benötigt wird.

Hierbei ist lediglich eine genaue Syntax zu beachten und dass die neuen Objekte keine bereits vorhandenen Objektnummern haben.

Weiters muss man, wenn man einen Javascript zu einem Formularfeld oder einer Seite hängt, oder es als Dokumentenvorgang eintragen will auch die genaue Stelle kennen an der der Verweis auf das neue Javascript eingefügt werden muss. Auch der Schlüssel für die Aktion bei der das neue Javascript ausgeführt werden soll muss bekannt sein.

9. Dokumentensicherheit

Um dem Bearbeiter einen gewissen Grad an Sicherheit zu geben, gibt es in Adobe Acrobat mehrere Möglichkeiten ein Dokument zu schützen. Ein Dokument kann mittels 40 oder 128 Bit oder mittels der eingebauten Self Sign Sicherheit verschlüsselt, mit einem Passwort versehen oder unterschrieben werden. Mit Version 8.0 von Adobe Acrobat steht nur noch die 128 Bit Verschlüsselung zur Verfügung.

"*Although not in the list in the original PDF specification in the more recent documents this has been added to the list. This calls for the features that PDF supports for encrypting the file's content, for digital signatures and for controlled restriction of certain manipulations of the PDF, like printing.*" [17]

9.1. Bit Verschlüsselung

Bei der 40 oder 128 Bit Verschlüsselung wird ein Dokument verschlüsselt. Zusätzlich können gewisse Aktionen gesperrt werden:

40 Bit Verschlüsselung:

- Drucken nicht zulässig
- Dokumentenänderungen nicht zulässig
- Kopieren oder entnehmen von Inhalt nicht zulässig, Zugriff deaktiviert
- Hinzufügen oder ändern von Kommentaren und Formularfeldern nicht zulässig

128 Bit Verschlüsselung

- Inhaltszugriff für Sehbehinderte aktivieren
- Kopieren und entnehmen von Inhalt zulassen
- Zulässige Änderungen:
    - Keine
    - Nur Dokumentenzusammenstellung
    - Nur Ausfüllen oder Unterschreiben von Formularfeldern
    - Kommentarerstellung, Ausfüllen oder Unterschreiben von Formularfeldern
    - Allgemeines Bearbeiten, Kommentieren und Erstellen von Formularfeldern
- Drucken
    - Nicht zulässig
    - Niedrige Auflösung
    - Zulässig

Diese Einstellungen sind jedoch nur sinnvoll wenn das Dokument mit einem „Viewer" geöffnet wird. In einem solchen Programm, das nur das Ansehen eines Dokuments erlaubt, ist es nicht möglich die Sicherheitseinstellungen zu ändern. Wird jedoch Acrobat Adobe verwendet, so sind dieses Einstellungen unnötig, da sie von dem Bearbeiter geändert werden können.

9.1.1. Verwendung von Passwörtern

Im Zuge einer Verschlüsselung können zwei verschiedene Passwörter vergeben werden:

- Kennwort zum Öffnen der Datei erforderlich (Benutzerkennwort)
- Kennwort zum Ändern von Berechtigungen und Kennwörtern erforderlich (Hauptkennwort)

Mit dem Benutzerkennwort wird das Dokument allgemein gesperrt. Wird jedoch kein Hauptkennwort vergeben, so ist es jemandem mit einem Benutzerkennwort möglich alle Kennwörter zu ändern oder sogar die Verschlüsselung aufzuheben.

Wird nur ein Hauptkennwort vergeben, so ist es für jeden möglich das Dokument mit den eingestellten Berechtigungen zu öffnen, jedoch ist es nicht möglich die Sicherheitseinstellungen zu ändern.

9.1.2. Verschlüsselungsart

Je nach Version von Acobat Adobe können verschiedenen Verschlüsselungsalgorithmen verwendet werden. Mit der Version Acrobat Adobe 8.0 (genauer mit Version 7.0 und höher) wird jedoch die Verschlüsselung mit dem 128-Bit AES Algorithmus und RC4 128-Bit unterstützt.

9.1.2.1. AES Verschlüsselung

Der Advanced Encryption Standard (AES) ist ein symmetrisches[4] Kryptosystem, das als Nachfolger für DES bzw. 3DES im Oktober 2000 vom National Institute of Standards and Technology (NIST) als Standard bekannt gegeben wurde. Nach seinen Entwicklern Joan Daemen und Vincent Rijmen wird er auch *Rijndael*-Algorithmus genannt. [30]

- **Arbeitsweise**: Rijndael ist ein Blockchiffre[5]. Bei Rijndael können Blocklänge und Schlüssellänge unabhängig voneinander die Werte 128, 160, 192, 224 oder 256 Bits erhalten, während bei AES die Einschränkung der festgelegten Blockgröße von 128 Bit und der Schlüsselgröße von 128, 192 oder 256 Bit gilt. Jeder Block wird zunächst in eine zweidimensionale Tabelle mit vier Zeilen geschrieben, deren Zellen ein Byte groß sind. Die Anzahl der Spalten variiert somit je nach Blockgröße von 4 (128 Bits) bis 8 (256 Bits). Jeder Block wird nun nacheinander bestimmten Transformationen unterzogen. Aber anstatt jeden Block einmal mit dem Schlüssel zu verschlüsseln, wendet Rijndael verschiedene Teile

---

[4] Ein **symmetrisches Kryptosystem** ist ein Kryptosystem, welches im Gegensatz zu einem asymmetrischen Kryptosystem den gleichen Schlüssel zur Ver- und Entschlüsselung verwendet. [6]
"*Asymmetric cryptography algorithms are often based on modular arithmetic, and operate on huge integers (512- 2048 bits). They require more processing power (due to modular exponentiation) than symmetric algorithm for an equivalent robustness. Moreover, ciphered text is longer than the original clear text; larger memories are thus needed.*" [20]

[5] Die **Blockverschlüsselung**, auch *Blockchiffre*, ist ein Algorithmus der einen Datenblock von gewöhnlich 64 oder 128 Bit mittels eines Schlüsselwerts verschlüsselt. Der verschlüsselte Block hat dabei die gleiche Länge. Typische Werte für die Schlüsselllänge moderner Verfahren sind 112, 128 und 168 Bit. [7]
Let M be a plaintext message. A block cipher breaks M into successive blocks M1, M2 . . . . . and enciphers each M with the same key K; that is,
EK(M) = EK(M1)EK(M2) . . . [18].

des erweiterten Originalschlüssels nacheinander auf den Klartext-Block an. Die Anzahl *r* dieser Runden variiert und ist von der Schlüssellänge *k* und Blockgröße *b* abhängig (beim AES also nur von der Schlüssellänge): [5]

Anzahl der Runden bei Rijndael. (Die für AES relevanten Werte sind farbig unterlegt.)

	***b* = 128**	***b* = 160**	***b* = 192**	***b* = 224**	***b* = 256**
***k* = 128**	10	11	12	13	14
***k* = 160**	11	11	12	13	14
***k* = 192**	12	12	12	13	14
***k* = 224**	13	13	13	13	14
***k* = 256**	14	14	14	14	14

Tabelle 6: Anzahl der Runden bei Rijndael. (Die für AES relevanten Werte sind farbig unterlegt.)

- Ablauf:
  - Schlüsselexpansion
    - Vorrunde
      - KeyAddition (Rundenschlüssel[0])
  - Verschlüsselungsrunden (wiederhole solange runde < R)
    - Substitution()
    - ShiftRow()
    - MixColumn()
    - KeyAddition(Rundenschlüssel[runde])
  - Schlussrunde
    - Substitution()
    - ShiftRow()
    - KeyAddition(Rundenschlüssel[R])

- Schlüsselexpansion: Zunächst muss der Schlüssel in R + 1 Teilschlüssel (auch Rundenschlüssel genannt) aufgeteilt werden. Die Rundenschlüssel müssen die gleiche Länge wie die Blöcke erhalten. Somit muss der Benutzerschlüssel auf die Länge b * (r + 1) expandiert werden, wobei b die Blockgröße angibt. Die Schlüssel werden wieder in zweidimensionalen Tabellen mit vier Zeilen und Zellen

der Größe 1 Byte gebildet. Die ersten Spalten der Tabelle werden mit dem Benutzerschlüssel gefüllt. Die weiteren Spalten werden wie folgt rekursiv berechnet: Um die Werte für die Zellen in der nächsten Spalte zu erhalten, wird zunächst das Byte, welches sich in der letzten (je nach Blockgröße: vierten, sechsten oder achten) Spalte befindet und drei Zeilen zurückliegt, durch die S-Box verschlüsselt und anschließend mit dem um eine Schlüssellänge zurückliegenden Byte XOR verknüpft. Befindet sich das zu berechnende Byte an einer Position, die ein ganzzahliger Teiler der Schlüssellänge ist, so wird das berechnete Byte zusätzlich mit einem Eintrag aus der rcon-Tabelle XOR verknüpft. Hierfür wird als Index eine fortlaufende Nummer verwendet. Die rcon-Tabelle ist ähnlich wie die S-Box eine Tabelle in Form eines Arrays, das konstante Werte enthält die auf einem mathematischen Zusammenhang beruhen. Jedes weitere Byte (das sich nicht in der ersten Spalte befindet) wird aus einer XOR-Verknüpfung des vorherigen Bytes (w_{i-1}) mit dem Byte einer Schlüssellänge vorher ($w_{i-k/32}$) gebildet.[5]

9.1.2.2. RC4 Verschlüsselung

Bei der RC4 Verschlüsselung handelt es sich um eine Stromchiffrierung[6] welche von Ronald L. Rivest (RC4 = **R**ons **C**ode **4**) entwickelt wurde.

„Bei RC4 gibt es eine S-Box S (Substitutionstabelle) mit 256 Einträgen der Länge 8 Bit (man sagt auch, die S-Box hat eine Größe von 8 · 8, d. h. 8 Bit werden ersetzt, 8 Bit erhält man als Ausgabe). Die Einträge in dieser Box reichen von S_0 (1. Eintrag) bis S_{255} (256. Eintrag). Außerdem gibt es zwei 8-Bit große Zähler i und j, die zu Anfang auf 0 gesetzt werden.“ [8]

Vor der Durchführung der RC4 Verschlüsselung muss die S-Box erstellt werden:

Zuerst wird die S-Box linear aufgefüllt: $S_0 = 0$, $S_1 = 1$, … $S_{255} = 255$. Dann wird folgende Schleife durchlaufen:

für *i* von 0 bis 255
  $j = (j + S_i + K_{i \bmod k}) \bmod 256$
  vertausche S_i und S_j
nächstes *i*

Wobei K der Schlüssel und *k* die Schlüssellänge ist. Falls der Schlüssel nicht genau 256 Byte lang ist, muss er wiederholt werden, man fängt also wieder bei K_0 an. Dies kann man einfach erreichen, indem man eine modulo-Operation durchführt, d. h. *i* mod Schlüssellänge.[31]

Im nächsten Schritt wird eine Zufallszahl erzeugt, die dann XOR mit dem Klartext verknüpft wird, und somit entsteht der Geheimtext.

1. Erhöhe *i* um 1
2. Addiere S_i zu *j*
3. Vertausche S_i und S_j
4. Setze die temporäre Variable *t* auf $S_i + S_j$.
5. Gebe S_t aus

Bei der Entschlüsselung, wird das Geheimtext Byte mit der Geheimzahl XOR verknüpft, und man erhält nun wieder das Klartext Byte.

---

[6] Bei der Stromchiffre wird der Geheimtext, welcher die gleiche Länge wie der Klartext hat, mittels eines Verschlüsselungsstroms verschlüsselt. Die Verschlüsselung erfolgt nicht blockweise sondern Zeichenweise. Die Zeichen des Klartextes werden mit dem Verschlüsselungsstrom mit XOR verknüpft.

9.2. Self-Sign Sicherheit

Die Self-Sign Sicherheit ermöglicht es dem Urheber eines .pdf Dokuments das Dokument für bestimmte Benutzer freizugeben. Voraussetzung ist, dass der Empfänger der Datei auch über ein Self-Sign Konto verfügt und dem Urheber des Dokuments die Schlüsseldatei des Empfängers vorliegt.

Durch diese Schlüsseldatei wird der Empfänger zu einer Liste von berechtigten Benutzern hinzuzufügen. Weiters können jedem einzelnen Benutzer Rechte für das Dokument zuzuordnen werden.

Da es sich auch bei der Self-Sign Sicherheit um einer 128 Bit Verschlüsselung handelt können hier dieselben Rechte wie in Abschnitt 9.1 beschrieben zugewiesen werden.

Um die Self-Sign-Sicherheit zu benutzen, muss „Werkzeug / Self-Sign-Sicherheit / Anmelden" geöffnet werden. Wenn noch kein Profil vorhanden ist, soll dieses neu erzeugt werden. Dazu muss mindestens der Name des Benutzers und ein Passwort mit mindestens 6 Zeichen eingegeben werden.

Danach kann unter „Werkzeug / Self-Sign-Sicherheit / Benutzereinstellungen…" unter „Benutzerinformationen" mit der Schaltfläche „In Datei exportieren…" eine .fdf Datei erzeugt werden. Diese .fdf Datei kann nun an einen anderen Benutzer gesendet werden.

Hat dieser Benutzer die .fdf Datei erhalten muss diese Datei unter „Werkzeug / Self-Sign-Sicherheit / Benutzereinstellungen…" in dem Bereich „Bestätigte Zertifikate" hinzugefügt werden.

Nun wird unter „Datei / Sicherheitseinstellungen…" die Self-Sign-Sicherheit gewählt. Jetzt kann unter allen bestätigten Zertifikaten gewählt werden, wer Zugang zu diesem Dokument erhalten soll. Weiters ist es noch möglich das Dokument für jeden Benutzer mit anderen Rechten zu versehen.

Wird nun das Dokument geöffnet erhält der Benutzer die Möglichkeit unter seinen Self-Sign-Anmeldungen zu wählen oder eine dementsprechende Datei zu suchen. Wenn die Anmeldung mit Passwort erfolgt ist, wird das Dokument wenn es dem Benutzer erlaubt ist, mit seinen Rechten geöffnet.

9.3. Unterschriftsfelder

"*An important feature of digital signatures is to serve as nonrepudiation evidence. To be eligible as non-repudiation evidence, a digital signature on an electronic document should remain valid until its expiry date which is specified by some non-repudiation policy. As signature keys may be compromised and the validity of signatures may become questionable, additional security mechanisms need to be imposed on digital signatures.*"[26]

Unterschriftsfelder an sich implizieren keine Verschlüsselung des Codes sondern dienen dazu in einem Dokument festzustellen ob sich seit einer Unterschrift etwas geändert hat. Wobei auch Unterschriftsfelder nur mit dem der Adobe Acrobat angebotenen Self-Sign Sicherheit, genauer mit den damit erstellten Konten, arbeitet.

Wird ein Dokument nach der Unterschrift verändert, so scheint dies in der Unterschriftenhistorie auf. Weiters besteht hier auch die Möglichkeit das momentane Dokument mit dem zum Zeitpunkt der Unterschrift zu vergleichen.

Das Unterschriftenfeld kann folgende Ausprägungen haben:

- Unterschrift komplett gültig:

  Hierbei kann man davon ausgehen, dass das Dokument völlig unverändert ist.

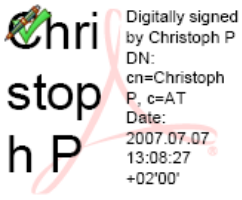


Abbildung 32: Unterschrift gültig

- Unterschrift teilweise gültig:

  Hier ist zwar die Unterschrift gültig, aber es kam zu Veränderungen des Dokumenteninhaltes. Dabei ist es unwichtig ob die Unterscheidung zu dem Ursprungsdokument in dem Text des Dokumentes an sich liegt, oder sich das Dokument in dem Inhalt eines Formularfeldes geändert hat.

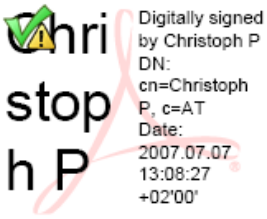


Abbildung 33: Unterschrift teilweise gültig

Ist eine Unterschrift nur teilsweise gültig können durch einen Klick auf das Feld Unterschriftenfeld folgende Optionen gewählt werden:

- Unterschriebene Version ansehen:

  Hierbei wir das Dokument in dem Zustand geöffnet in welchem es unterschrieben wurde.

- Versionen vergleichen:

  Wird diese Option gewählt, wird das vorhandene Dokument und auch das unterschriebene Dokument geöffnet und auf deren Aussehen überprüft. Diese Dokumente werden gemeinsam angezeigt, und es werden alle Veränderungen in dem Dokument farblich hervorgehoben.

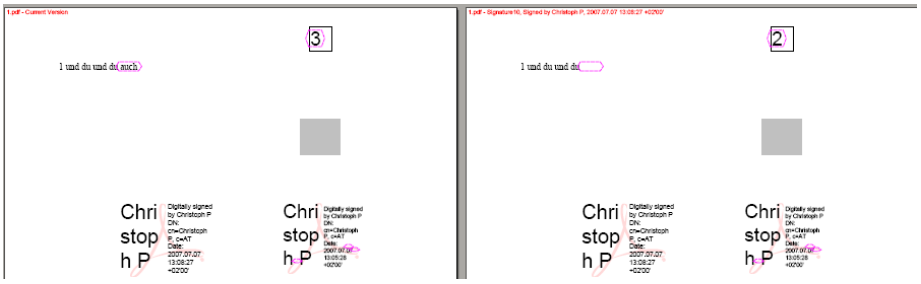


Abbildung 34: Dokumentenvergleich bei Unterschrift

9.4. Zusammenfassung

Sobald ein Dokument verschlüsselt ist, egal ob mit 40 oder 128 Bit Verschlüsselung oder mittels Self-Sign Verschlüsselung, kann der Code eines Dokuments nicht mehr verändert werden.

Durch die Verschlüsselung enthält der Code die Objektreferenzen nicht in Klartext. Dadurch kann nicht herausgefunden werden, welches Objekt für welchen Aufruf verantwortlich ist. Weiters sind auch die Objekte welchen den Javascriptcode für Vorgänge enthalten verschlüsselt. Es ist also auch nicht möglich ein vorhandenes Javascript abzuändern.

Weiters kann das Dokument sobald der Code verändert wurde (auch bei einfügen eines Leerzeichens) nicht mehr in Acrobat Adobe geöffnet werden. Es erscheint eine Fehlermeldung und das Dokument wird geschlossen. Natürlich ist es weiterhin möglich das Dokument mittels des internen Javascripteditors zu ändern, sofern die Rechte dazu bestehen.

Bei Dokumenten mit Unterschriftsfeldern verhält sich das etwas anders. Der Code wird weiterhin in Klartext angezeigt und kann auch verändert werden. Dies gestaltet sich jedoch außerordentlich schwierig da es im Code zu einer komplexen Historisierung, über das Maße dessen hinaus was in Abschnitt 8.3 beschrieben wird. Dadurch kommt es bei Neueinführung von Javascript durch Codeveränderungen sehr leicht zu Fehlern, durch welche sich das Dokument dann nicht öffnen lässt.

Auch hier wird in der Unterschriftenhistorie angezeigt, dass das Dokument verändert wurde.

Wird ein Dokument schon vor der Erstellung mit einem Javascript ausgestattet, welches zum Beispiel den Inhalt eines Textfeldes verändert, so wird hier auch eine Änderung des Dokumenteninhaltes in der Unterschriftenhistorie angezeigt. Ist der Text in einem Dokument verändert, so wird dies auch in der Unterschriftenhistorie angezeigt. Außerdem ist es jederzeit möglich den Unterschriftenzustand eines Dokumentes wieder herzustellen, oder sich alle Veränderungen zwischen dem momentanen und dem unterschriebenen Dokument grafische anzeigen zu lassen.

Wenn ein Dokument also mit einer Unterschrift versehen ist, so ist es weder direkt noch über ein Javascript möglich das Dokument zu verändern ohne dass der Benutzer dies sieht und die Änderung auch nachvollziehen kann.

10. Acrobat Versionen und pdf Spezifikationen

Wie bei jeder Software ändert sich mit einer neuen Version nicht nur die „Versionsnummer" eines Programms (Adobe Acrobat 6, Adobe Acrobat 7, …) sondern auch der zugrunde liegende Code. Die Spezifikation des Codes kann in dem .pdf Code in der ersten Zeile gelesen werden.

```
%PDF-1.6
%âãÏÓ

14 0 obj <</Linearized 1/L 7897/O 17/E 2296/N 1/T 7555/H [ 516
164]>>
endobj


xref

14 10
```

Die Versionen entsprechen den folgenden Codespezifikationen in Tabelle 7: Acrobat Versionen.

Acrobat Version	Pdf Version	Veröffentlichung
Version 1	1.0	Juni 1993
Version 2	1.1	September 1994
Version 3	1.2	Juli 1996
Version 4	1.3	April 1999
Version 5	1.4	Mai 2001
Version 6	1.5	April 2003
Version 7	1.6	Januar 2005
Version 8	1.7	2007

Tabelle 7: Acrobat Versionen

In ihrer Grundfunktion sind alle Acrobat Versionen abwärts kompatibel (es können Dokumente die mit alten Versionen erstellt wurden mit jeder darauf folgenden geöffnet werden).

Wird ein Dokument in einer höheren Version erstellt, kann es zu Problemen bei Funktionen kommen, die in niedrigeren Versionen noch nicht implementiert waren. Das Ansehen eines Dokuments welches lediglich Bilder und Text enthält ist jedoch gewährleistet. Weiters bietet Acrobat bei wichtigen Funktionen wie der Sicherheit die Möglichkeit ein Dokument für eine niedrigere Version zu speichern.

Bei Acrobat selbst gibt es die Spezifikationen für den pdf Code erst ab der Version 1.4.

Im Folgenden wird auf die Spezifikation des Codes 1.6 eingegangen, welcher in der Version 7 verwendet wird.

10.1. Flatedecode

Wenn man sich den Codes eines .pdf Dokumentes ansieht, und den Objekten folgt, so wird man irgendwann zu dem „Content" einer Seite kommen. Dieses Objekt unterscheidet sich aber von denen die bis jetzt bekannt waren.

Der Seiteninhalt einer Seite wird bei Acrobat Adobe (ab Version 1.2) komprimiert, um das .pdf Dokument kleiner zu machen.

Für diese Komprimierung, wir ein Verfahren verwendet, welches von Phil Katz erfunden wurde. Dieses Verfahren wird auch in PNG und TIFF Grafiken sowie bei der Erstellung von .zip oder .cab Files eingesetzt.

Dieses Verfahren ist auf der Homepage www.zlib.net frei als Source und auch als kompilierte DLL verfügbar.

Um nun den Inhalt einer Seite aus einem .pdf Dokument lesen zu können, ohne einen entsprechenden .pdf Reader zu haben, muss der Inhalt der Seite von einem Programm mittels „Flateencode" decodiert bzw. entpackt werden.

10.1.1. Der Deflate Algorithmus

Der Deflate Algorithmus setzt sich aus 2 Komponenten zusammen. Einerseits aus dem Huffman Code und andererseits aus der LZ77 Komprimierung. [21]

- Huffman-Code: Dieser Algorithmus ist eine Erweiterung bzw. Verbesserung des Shannon-Fano-Algorithmuses, da es Huffman schafft einen binären Baum zu konstruieren der immer die optimale Form für die gegebenen Wahrscheinlichkeiten erzeugt. [10]
- LZ77: Hierbei handelt es sich um ein gleitends Fenster (engl.: sliding window) Algorithmus, welcher ein Fenster von konstanter Größe unterteilt in Vorschau-Puffer und Textfeld besitzt. Dieses Fenster wir nun über den zu komprimierenden Text bewegt, wobei sich in dem Vorschau-Puffer der Text befindet, welcher komprimiert werden soll. [10]

  In dem Textfeld befinden sich bereits komprimierte Zeichenketten, die für die weitere Komprimierung als „Lexikon" dienen.

Bei der Anwendung des Deflate Algorithmus werden zuerst die durch Benutzung des LZ77 Algorithmus mehrfach vorkommende Textpassagen ersetzt. Danach kommt es zu einer Entropiekodierung nach Huffman.[10]

10.1.2. Flateencode Programm für pdf

Bei dem folgenden Programm handelt es sich um eine Windows Applikation, welche in Microsoft Visual Studio.Net in der Sprach C# entwickelt wurde.

```
Using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using ICSharpCode.SharpZipLib.Zip.Compression.Streams;

namespace flatedecode
{
    class Program
    {
        static void Main(string[] args)
        {
            //Mit Einagbe der Hex
            byte[] bb = new byte[] {
```

```
0x48, 0x89, 0x14, 0x8D, 0xBD, 0x0A, 0xC2, 0x50, 0x14, 0x83, 0xF7,
0xF3, 0x14, 0x19, 0x75, 0xB9, 0x37, 0x47, 0xA1, 0x75, 0x28, 0x1D,
0xFA, 0x83, 0x38, 0x14, 0x04, 0xF3, 0x02, 0x22, 0xF8, 0x47, 0x45,
0xB0, 0x83, 0xE0, 0xD3, 0x7B, 0xEE, 0x92, 0x2F, 0x84, 0x84, 0xE4,
0x23, 0x9A, 0x26, 0x4F, 0xFD, 0x61, 0x00, 0xD1, 0xB6, 0xDD, 0xD0,
0xC3, 0x88, 0x9B, 0x39, 0x1E, 0xB0, 0xBC, 0x3F, 0x85, 0x5F, 0xAC,
0x93, 0x65, 0x89, 0x70, 0xE8, 0x6A, 0x4C, 0xA4, 0x87, 0xBB, 0xC4,
0x40, 0xDF, 0x10, 0x68, 0x81, 0xB3, 0xF0, 0x57, 0xA2, 0x0F, 0x7C,
0x13, 0x64, 0x41, 0xCD, 0xB4, 0xAB, 0x50, 0x57, 0x4C, 0x51, 0xD8,
0x42, 0x2F, 0x5B, 0xDD, 0xCF, 0xF3, 0xFC, 0xC6, 0x5A, 0x4F, 0x1B,
0x65, 0xE3, 0x14, 0x77, 0x7F, 0x01, 0x06, 0x00, 0x43, 0xCD, 0x1C,
0x8D};
            MemoryStream ms = new MemoryStream(bb);
            ms.ReadByte();
            ms.ReadByte();
            System.IO.Compression.DeflateStream ds = new
             System.IO.Compression.DeflateStream(ms,
             System.IO.Compression.CompressionMode.Decompress);
            Console.WriteLine(new StreamReader(ds).ReadToEnd());
            Console.Read();
        }
}
```

Code 21: Deflate Programm

In diesem Programm werden in das Byte Array bb die Bytes des zu dekomprimierenden Strings geschrieben. Danach wird ein neuer DeflateStream des erzeugt und per Parameter „Decompress" der Dekomprimierungsmodus ausgewählt. Danach wird dem DeflateStream ds das Byte Array bb zur Dekomprimierung übergeben.

Wichtig ist hierbei, dass bei einer .pdf Date der Stream noch einen Header von 2 Byte hat. Diese 2 Byte müssen beim Lesen übersprungen werden. Durch diese 2 Byte Header ist es auch nicht möglich einen Stream einfach zu ändern und wieder in die .pdf Datei zu spielen.

Im Folgenden ist beschrieben wie man zu dem benötigten Stream gelangt.

Die Abbildung 35 zeigt das Dokument wenn es mit Adobe Acrobat geöffnet wird.

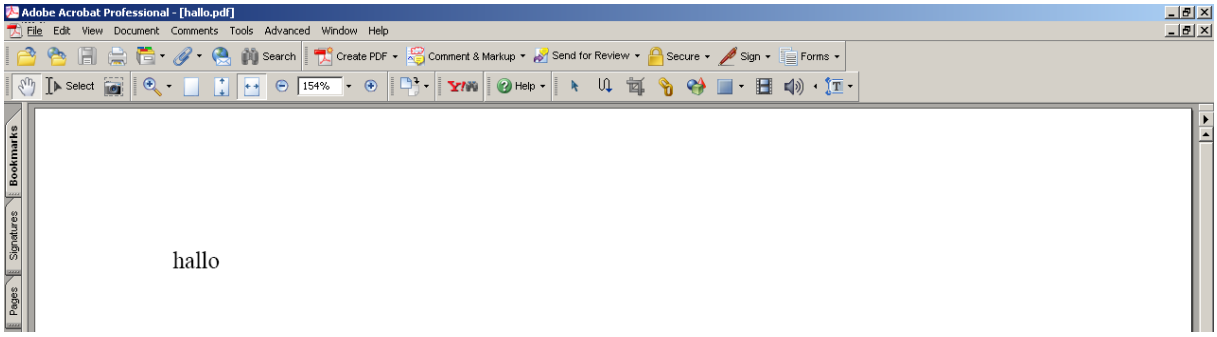


Abbildung 35: Seite in Acrobat(hallo)

Wie in Code 22 zu sehen ist, muss man den den Objekten folgen, bis man zu dem Stream der die gewünschten Seite beschreibet, gelangt.

```
<</Size 23/Prev 7455/XrefStm 639/Root 15 0 R/Info 6 0
R/ID[<D2328A400B7E0568EBFCF41A7336C4D4><86142D3037B4BC41A002C524319
5E903>]>>

Startxref

                                    ……

15 0 obj<</MarkInfo<</LetterspaceFlags 0/Marked true>>/Metadata 5 0
R/PieceInfo<</MarkedPDF<</LastModified(D:20070618121912)>>>>/Pages
4 0 R/PageLayout/OneColumn/StructTreeRoot 7 0
R/Type/Catalog/LastModified(D:20070618121912)/PageLabels 2 0 R>>
endobj

                                    ……

4 0 obj<</Count 1/Type/Pages/Kids[16 0 R]>>
endobj

                                    ……

16 0 obj<</CropBox[0 0 595.22 842]/Parent 4 0 R/StructParents
0/Contents 17 0 R/Rotate 0/MediaBox[0 0 595.22
842]/Resources<</Font<</TT0 18 0
R>>/ProcSet[/PDF/Text]/ExtGState<</GS0 20 0 R>>>>/Type/Page>>
endobj

                                    ……

17 0 obj<</Length 144/Filter/FlateDecode>>stream

H‰ □½

ÂP ƒ÷ó u¹7G¡u( úƒ8 ó ”øGE°ƒàÓ{î’/„„ä#š&Oýa
Ñ¶ÝÐÃˆ›9-°¼?…_¬“e‰pèjL¤‡»Ä@ß h□³ðW¢□|□dAÍ´«PWLQØB/[ÝÏóüÆZO□eã□
w□□□ CÍ□□

endstream
endobj
```

Code 22: Code des .pdf (hallo)

In der Abbildung 36 ist ersichtlich, dass das in Code 22 gezeigte Objekt 17 genau jenen Bytes entspricht die in dem Code 21 dem DefalteStream zur Dekomprimierung über geben wurden.

Weiter ist hier zu sehen, dass der Stream von den Bytes 0A und 0D am Anfang als auch am Ende begrenzt ist. Diese Bytes werden nicht in den Code 21 übernommen.

Will man jedoch den in Code 21 beschrieben Code erweitern um das Auslesen zu automatisieren, können diese 2 Bytes als Suchkriterium in dem .pdf Code verwendet werden. Hierbei müsste immer der Code zwischen den beiden Bytes (ohne diese) ausgelesen und dekomprimiert werden.

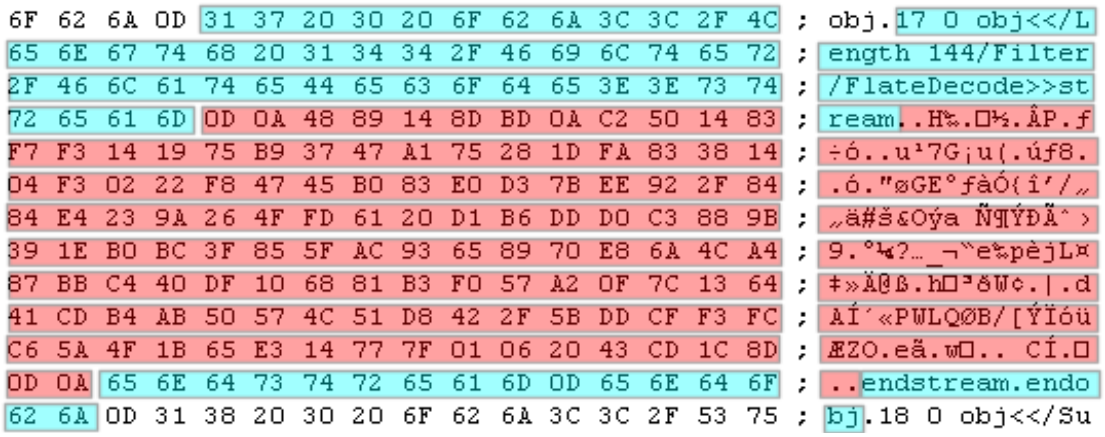


Abbildung 36: Hexadezimalcode des .pdf (hallo)

Wird nun der Code 21 ausgeführt so kann man den Stream in der Konsole als Klartext ablesen. Folgende Informationen sind in dem Klartextcode vorhanden:

Key	Verwendung
/p << / MCID 0>>	Eine Identifizierung, dass hier ein grafischer Bereich angekündigt wird. (Auch Text wird im weitesten Sinne nur als Grafik angesehen)
BDC	Hier beginnt der grafische Bereich
G	Dieser Schlüssel beschreibt den Grauton
i	Beschreibt die „Flatness Tolerance". Dies beschreibt bei einer Grafik, wie weit der Pixel sich von der optimalen Position entfernen darf. Ein Wert von 0 bis 100 kann eingetragen werden, wobei ein Wert von 0 die geringste Toleranz zulässt.
Gs	Hier wird der Name für das verwendete Grafikwörterbuch angegeben.
BT	Hier beginnt das Textobjekt
Tf	Beschreibt die Schriftart und auch die Größe
Tc	Beschreibt den Abstand zwischen den Buchstaben
Tw	Beschreibt den Abstand zwischen den Wörtern
Ts	Wird benutzt um Text hoch oder tief zu stellen
Tr	Wird benutzt um die Darstellung der Buchstaben zu verändern.
Tj	Zeigt den Text
ET	Ende des Textobjektes
EMC	Ende des grafischen Bereiches

Tabelle 8: Schlüssel des Grafikobjektes [23]

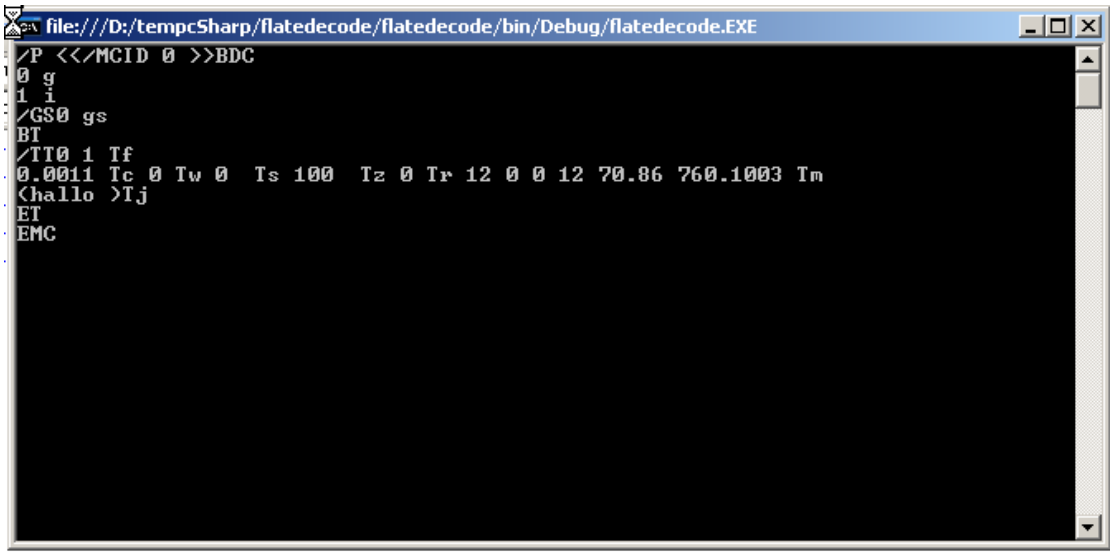


Abbildung 37: Ergebnis des Deflate Programms

10.2. Passwortsicherung in Acrobat Adobe 7

Im Folgenden wird beschrieben wie die Passwörter erstellt werden, und wie sie in einem .pdf Dokument gespeichert werden:

10.2.1. Allgemein

Grundsätzlich gibt es 2 Varianten um in Acrobat Adobe ein Passwort zu vergeben.

- Owner Password: Dieses Passwort wird benutzt um das Öffnen eines Dokumentes zu verbieten.
- User Password: Dieses Passwort wird verwendet um die Informationen bezüglich Editierung und Druck zu verschlüsseln und auch um den Bereich der Sicherheitseinstellung zu schützen.

Diese Passwörter werden nicht nur für die Usereingabe, sondern auch für die Verschlüsselung der Informationen in dem .pdf Code benutzt.

Wenn man den Code eines .pdf Dokumentes öffnet, wie in Code 23 ersichtlich, sieht man, dass es sich um ein Dokument mit Sicherheit handelt. Es ist jedoch aufgrund der Verschlüsselung nicht erkenntlich ob es sich um ein Passwort zum öffnen (Owner Password) oder ein Passwort zum Schutz der Sicherheit innerhalb des Dokumentes handelt.

```
%PDF-1.6
%âãÏÓ

14 0 obj <</Linearized 1/L 7905/O 17/E 2303/N 1/T 7563/H [ 516
164]>>
endobj

xref

14 10

0000000016 00000 n

0000000845 00000 n

0000000981 00000 n

0000001245 00000 n

0000001456 00000 n
```

```
0000001647 00000 n
0000001984 00000 n
0000002227 00000 n
0000000680 00000 n
0000000516 00000 n
trailer
<</Size 24/Prev 7552/XrefStm 680/Root 16 0 R/Encrypt 15 0 R/Info 6
0
R/ID[<9C8437DE1CF198075208D713B331EDAB><102D2DC895BE93479A845F85241
BCB62>]>>
```

Code 23: Encryption Verweis

In Code 23 ist gut zu sehen, dass sich hier das „Encryption" Tag befindet, aus welchem ersichtlich ist, dass das Dokument verschlüsselt ist. Hier wird auf ein weiteres Objekt (hier das Objekt 15) verwiesen.

In dem Objekt 15 finden sich Optionen zur Verschlüsselung:

```
15          0          obj<</Length          128/Filter/Standard/O(-
 ÿÏÓ¢□£Ú^Í•Ôê0c\rŒtÂÅÒ=Ýüà®Y†e□)/P                          -3392/R
3/U( îjˆW tQ¿ }ýû#îÍ™ÒÖ ¤ÒÖ ³ÒÖ ¿ÒÖ )/V 2>>
endobj
```

Code 24: Encryption Objekt

Die folgende Tabelle beschreibt die möglichen Ausprägungen des „Encryption" Objekts:

Schlüssel	Typ	Beschreibung
V	Nummer	Beschreibt grob den verwendeten Algorithmus 0. Ein Algorithmus welcher undokumentiert und nicht mehr unterstütz wird. Dieser Algorithmus sollte nicht mehr verwendet werden. 1. Ein Algorithmus mit der Länge von 40 Bit. 2. Ein Algorithmus der einen Schlüssel der Länge 40 Bit und mehr erlaubt 3. Ein nicht freigegebener Algorithmus der die Länge 40 bis 128 Bit erlaubt 4. Hier wird eine Algorithmus beschrieben, welcher sich auf die Werte CF, StmF und StrF bezieht.
Length	Nummer	Die Länge des Schlüssels für die Entschlüsselung. (vielfaches von 8)
CF	Dictionary	Ein Dictonary dessen Schlüssel die Namen und Werte des dazugehörigen Filter sind. Jeder Filter der nicht der Standardfilter ist, muss hier eingetragen sein.
StmF	Name	Der Name des Filters der verwendet wird. Dieser Name muss ein Schlüssel aus CF sein oder ein Standardfilter.
StrF	Name	Der Name des Filters der Benutzt wird, wenn alle "Strings" in dem Dokument verschlüsselt werden.
R	Nummer	Beschreibt welcher Standard „Securitc handler" verwendet werden soll. 2. Wird verwendet wenn das Dokument mit einem V Wert kleiner als 2 verschlüsselt wird und keine

		Rechte gesetzt sind. 3. Wird verwendet wenn das Dokument mit eine V Wert von 2 oder 3 verschlüsselt wurde und Rechte gesetzt sind. 4. Wenn das Dokument mit einen V Wert von 4 verschlüsselt wurde.
O	String	Dies ist ein 32- Bit String der mit dem Owner und User Passwort zusammenhängt und benutzt wird, um zu prüfen, ob eine richtiges Owner Passwort eingegeben wurde.
U	String	Dies ist ein 32- Bit String der mit dem User Passwort zusammenhängt und benutzt wird, um zu prüfen, ob ein richtiges User Passwort eingegeben wurde.
P	Nummer	Eine Nummer die aus einer 32-Bit Zahl entsteht wobei bei dieser Zahl für jedes Bit eine bestimmt Funktion hinterlegt ist. (siehe Tabelle 10: Rechtesystem)

Tabelle 9: Ausprägungen des Encryption Objekts [23]

Wie schon in Tabelle 9 beschrieben, werden über den Schlüssel P die Rechte in einem .pdf Dokument gesetzt. Hierbei handelt es sich um eine Zahl die in Bit umgerechnet wird, und somit für jedes einzelne Bit eine Funktion hinterlegt werden kann, welche mit 0 und 1 ein bzw. ausgeschaltet wird.

Wie in Tabelle 10 zu sehen ist, müssen die Bits 13 – 32 immer 1 sein. Um dies zu realisieren, wird diese Zahl P immer negativ angegeben.

Eine Veränderung dieser Zahl in dem .pdf Code, führt jedoch dazu, dass beim Öffnen des Dokuments, welches kein Passwort zum Öffnen hatte, nun ein Passwort verlangt wird.

Bit Nummer	Funktion
1	Reserviert (muss 0 sein)
2	Reserviert (muss 0 sein)
3	Dokument drucken (Qualität hängt von Flag 12 ab)
4	Dokument editieren. (Betrifft alle Funktionen die nicht in den Flags 6, 9 und 11 enthalten sind.)
5	Text und Grafiken aus dem Dokument kopieren oder extrahieren (außer Funktionen die in Flag 10 beschrieben sind.)
6	Anmerkungen hinzufügen und Ausfüllen von interaktiven Feldern. Wenn Flag 4 gesetzt ist auch das Erstellen und das Verändern von interaktiven Feldern.
7	Reserviert (muss 1 sein)
8	Reserviert (muss 1 sein)
9	Ermöglicht das Ausfüllen von bereit vorhandenen Formularfeldern
10	Gibt an ob es möglich ist, Text und auch Grafiken aus der .pdf Datei zu extrahieren.
11	Ermöglich das Anpassen des Dokuments (Rotieren, Einfügen oder Löschen von Seiten sowie das Erstellen von Bookmarks oder Vorschaubildern)
12	Gibt an wie das Dokument gedruckt werden kann. Dieses Flag steht im Zusammenhang mit dem Flag 3.

	Wenn dieses Flag und das Flag 3 gesetzt ist kann das Dokument in hoher Qualität gedruckt werden. Ist nur Flag 3 gesetzt kann das Dokument nur mit niedriger Qualität gedruckt werden.
13-32	Reserviert (muss 1 sein)

Tabelle 10: Rechtesystem [23]

10.2.2. MD5

Bei MD5 handelt es sich um eine kryptographische Hash-Funktion. Das Ziel einer Hash Funktion ist es aus einer Übermenge eine kleinere Untermenge zu bilden, und somit bei z.B. zwei langen Dokumenten nicht dieses Dokumente vergleichen zu müssen, sondern lediglich die Hash-Funktion um sagen zu können ob die Dokumente mit an Sicherheit grenzender Wahrscheinlichkeit gleich sind, oder sicher verschieden.

Eine weitere Eigenschaft von Hash Funktionen ist, dass Informationen bei der Erstellung des Hash verloren gehen, und der Algorithmus unidirektional funktioniert. Dadurch ist es zwar möglich, wenn man den Algorithmus kennt, aus einer Menge immer denselben Hash zu erzeugen, aber es ist unmöglich aus dem Hash die Ursprungsmenge wieder herzustellen.

Aus diesem Grund eignet sich der Hash Algorithmus auch dazu Passwörter zu vergleichen.

Bei dem MD5 Algorithmus handelt es sich um ein spezielles Verfahren, welches von Professor Ronald L. Rivest 1991 wegen der Unsicherheit des MD4 Algorithmus entwickelt wurde.

Es wird hierbei aus einer variablen Menge eine Ausgabe mit der Länge von 128 Bit generiert. Dazu wird an die Eingabemenge eine 64 Bit Zahl angehängt welche die Länge der Eingabemenge repräsentiert. Der Bereich zwischen Eingabemenge und 64 Bit Zahl wird mit einer Eins (1) und so vielen Nullen (0) aufgefüllt, bis die Gesamtmenge der Bits durch 512 teilbar ist.

MD5 arbeitet weiters mit einem 128-Bit-Puffer, welcher in vier 32-Bit Mengen unterteilt wird, welche mit Konstanten initialisiert werden. Mit diesem Puffer wird nun der erste 512-Bit Block verschlüsselt. Die Bearbeitung eines Blockes geschieht in 4 ähnlichen Runden. Jede Runde besteht aus 16 Operationen, welche eine Funktion benutzen die nicht linear ist und modularer Addition und Linksrotation.

Auf das Ergebnis wird dieselbe Funktion mit dem zweiten Parameterblock aufgerufen. Dies wir bis zum Ende der Eingabemenge durchgeführt[11]

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus, \wedge, \vee, \neg$ stehen jeweils für XOR, AND, OR und NOT-Operationen.

Abbildung 38: Funktion des MD5 Algorithmus

"*The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2]. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater*

likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard." [16]

10.2.2.1. Brute Force Attack

Hierbei handelt es sich um einer Angriffsmethode, welche auch bei einer MD5 Verschlüsselung angewandt werden kann.

Man nehme an, es handelt sich um ein Passwort, welches mit MD5 verschlüsselt wurde. Nun ist es nicht erreichbar das Passwort aus dem Hash zu generieren. Also muss der umgekehrte Weg genommen werden. Mann muss alle möglichen Kombinationen an Passwörtern ausprobieren. Dies kann bei langen Passwörtern zu einem unverhältnismäßigen Zeitaufwand führen.

Folgendes Beispiel zu Veranschaulichung des Zeitverbrauches.

Als erstes werden die möglichen Kombinationen errechnet:

1. Mögliche Zeichen $^{\text{Länge des Passwortes}}$ = Kombinationen
2. Kombinationen / Rechnergeschwindigkeit(Keys/sec) = Sekunden bis Fertigstellung
3. Es wird in dem folgenden Beispiel mit 30 Millionen Keys pro Sekunde gerechnet (= Durchschnitts PC 2004)

- Zahlen

Länge	Kombinationen	Sekunden	Minuten	Stunden	Tage	Jahre
11	100000000000	3317	55.28			
12	1000000000000	33168	552.80	9.21		
13	10000000000000	331677	5527.95	92.13	3.84	
14	100000000000000	3316772	55279.53	921.33	38.39	
15	1000000000000000	33167716	552795.26	9213.25	383.89	1.05

Tabelle 11: Brute Force Attack mit Zahlen

- Kleinbuchstaben

Länge	Kombinationen	Stunden	Tage	Jahre
11	3670344486987776	33815.82	1408.99	3.86
12	95428956661682170	879211.26	36633.80	100.37
13	2481152873203736600	22859492.65	952478.86	2609.53
14	64509974703297150000	594346808.85	24764450.37	67847.81
15	1.677259342285726e+21	15453017030.20	643875709.59	1764043.04

Tabelle 12: Brute Force Attack mit Kleinbuchstaben

- Alle Buschstaben

Länge	Kombination	Stunden	Tage	Jahre
11	7516865509350965000	69254794.30	2885616.43	7905.80
12	390877006486250200000	3601249303.35	150052054.31	411101.52
13	2.032560433728501e+22	187264963774.27	7802706823.93	21377278.97
14	1.0569314255388205e+24	9737778116261.88	405740754844.24	1111618506.42

15	5.496043412801867e+25	506364462045617.70	21098519251900.74	57804162333.97

Tabelle 13: Brute Force Attack mit allen Buchstaben

10.2.3. Das Ownerpasswort

Wie schon beschrieben, wird das Ownerpasswort dazu verwendet um das Dokument zu sichern. Dieses Passwort wird wie folgt erstellt:

1. Als Erstes muss das Passwort aus genau 32 Bytes bestehen. Um das zu realisieren, muss das Passwort entweder abgeschnitten oder mit den folgenden Bytes aufgefüllt werden :
   28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08
   2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A

2. Das Ergebnis aus Schritt 1 wird nun als Ausgangsmenge für den MD5 Algorithmus verwendet.

3. Wiederholung des zweiten Schrittes fünfzig (50) mal.

4. Nun wird das Ergebnis aus Schritt 3 mit einem RC4 Algorithmus verschlüsselt.

5. Das Ergebnis aus dem vierten Schritt wird nun wieder wie in Schritt 1 behandelt.

6. Erneute Verschlüsselung mit RC4

7. Nun wird das Ergebnis aus dem sechsten Schritt 19 mal mit RC4 verschlüsselt, wobei es zu einer XOR Verknüpfung des jeweiligen Ergebnisses aus dem sechsten Schritt, und einem einzelnen Byte aus dem vierten Schritt kommt.

8. Das Ergebnis aus dem siebenten Schritt wird als Ownerpasswort in dem .pdf Code gespeichert.

Dieser Wert, dient als Ausgangswert für weitere Verschlüsselungen innerhalb des .pdf Dokumentes

10.2.4. Das Userpasswort

Wie schon beschrieben, wird das Userpasswort dazu verwendet um das Dokument zu sichern. Dieses Passwort wird wie folgt erstellt:

1. Als erstes muss das Passwort (User Passwort) aus genau 32 Bytes bestehen. Um das zu realisieren, muss das Passwort entweder abgeschnitten oder mit den folgenden Bytes aufgefüllt werden:
   28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08
   2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A
2. Initialisierung der Hash Funktion mit dem Ergebnis aus dem ersten Schritt.
3. Nun wird auch der erste Teil der Dokumenten ID an die Hash Funktion übergeben.
4. Das Ergebnis aus der Hash Funktion wird mit dem Schlüssel aus dem ersten Schritt verschlüsselt.
5. Nun wird das Ergebnis aus dem vierten Schritt 19 mal mit RC4 verschlüsselt, wobei es zu einer XOR Verknüpfung des jeweiligen

Ergebnisses aus dem ersten Schritt, und einem einzelnen Byte aus dem vierten Schritt kommt.

6. Das Ergebnis aus dem siebenten Schritt wir als Userpasswort in dem .pdf Code gespeichert.

10.2.5. Erstellung eines allgemeinen Schlüssels

Hier wird ein allgemeiner Schlüssel erstellt welcher immer wieder in dem Dokument verwendet wird:

1. Als erstes muss das Passwort (User Passwort) aus genau 32 Bytes bestehen. Um das zu realisieren, muss das Passwort entweder abgeschnitten oder mit den folgenden Bytes aufgefüllt werden :
   28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08
   2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A

2. Initialisierung der Hash Funktion mit dem Ergebnis aus dem ersten Schritt

3. Hinzufügen des Ergebnisses aus Abschnitt 10.2.3 zu dem Hash.

4. Hinzufügen des P Wertes (siehe Abschnitt 10.2) zu dem Hash

5. Nun wird auch der erste Teil der Dokumenten-ID an die Hash Funktion übergeben.

6. Wenn die Metadaten des Dokuments nicht verschlüsselt werden sollen müssen noch 4 Byte mit dem Wert 0xFFFFFF übergeben werden.

7. Nun wird das Ergebnis aus dem Hash als Eingangswert benutzt und 50 mal gehasht.

8. Das Ergebnis ist der „encrypton key".

10.2.6. Private-Public-Key

„In public key cryptography each user or the device taking part in the communication have a pair of keys, a public key and a private key, and a set of operations associated with the keys to do the cryptographic operations. Only the particular user/device knows the private key whereas the public key is distributed to all users/devices taking part in the communication. Since the knowledge of public key does not compromise the security of the algorithms, it can be easily exchanged online. "[32]

Wie schon in Abschnitt 9.2 beschrieben, ist es auch möglich ein .pdf Dokument mit einem Privat-Public Key System zu verschlüsseln. Wird dies gemacht, so geschieht dies in dem folgend beschriebenen Verfahren.

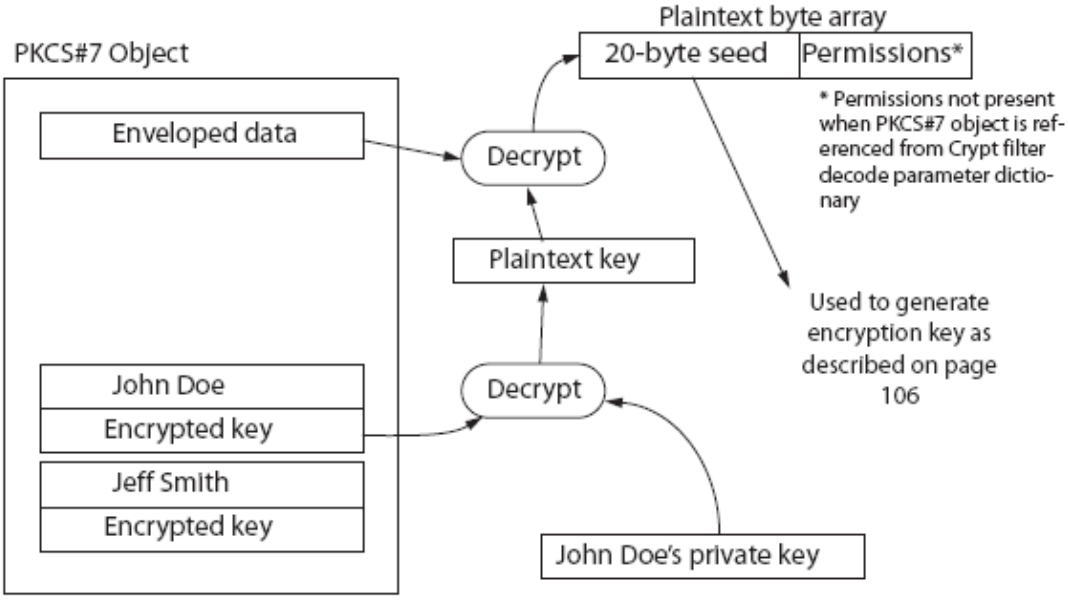


Abbildung 39: Public-Private Key[23]

Die Daten in dem PKCS#7 Objekt enthalten Schlüssel welche benutzt werden müssen, um das Dokument, oder genauer die verschlüsselten Teile des Dokumentes zu entschlüsseln. Für diese Entschlüsselung wird ein Schlüssel benutzt.
Dieser Schlüssel welcher in der Abbildung 39 als „Plaintext key" zu sehen ist, wird für jeden Benutzer des Dokuments mit Hilfe seines öffentlichen Schlüssels erzeugt, und in dem PKCS#7 Objekt gespeichert.
Um das Dokument wieder zu entschlüsseln, wird der private Schlüssel verwendet um das PKCS#7 Objekt zu entschlüsseln. Durch die Entschlüsselung des PKCS#7 Objekts enthält man ein Bytearray welches folgende Informationen liefert:

- Ein 20 Byte langes Array das für die Verschlüsselung der zu verschlüsselnden Objekte mit RC4 oder AES verwendet wird.
- Ein 4 Byte Array welches die Rechte in dem Dokument für diesen Benutzer angibt.

Durch dieses System ist es auch möglich für dasselbe Dokument, welches unter Umständen an verschiedenen Benutzer gesendet oder geliefert wird verschiedene Rechte (pro Benutzer) zu vergeben.

10.2.7. Unterschriftsfelder

Wie schon aus Abschnitt 9.2 bekannt, ist es möglich ein Dokument mittels eines Unterschriftenfeldes zu sichern. Hierbei wird im .pdf Code wie bereits bekannt ein neues Element vom Typ Unterschriftenfeld hinzugefügt.

```
</Size 36/Prev 14984/XrefStm 892/Root 18 0 R/Info 9 0
R/ID[<9C8437DE1CF198075208D713B331EDAB><D3A0946A79A2F3448089B153660
1D179>]>>

                                    ……

18 0 obj<</MarkInfo<</LetterspaceFlags 0/Marked true>>/Metadata 8 0
R/AcroForm                          19                              0
R/PieceInfo<</MarkedPDF<</LastModified(D:20070708113719)>>>>/Pages
7      0      R/PageLayout/OneColumn/StructTreeRoot      10      0
R/Type/Catalog/Lang(DE)/LastModified(D:20070708113719)/PageLabels 5
```

```
0 R>>
endobj

                                        ……

7 0 obj<</Count 1/Type/Pages/Kids[20 0 R]>>
endobj

                                        ……

20 0 obj<</CropBox[0 0 595.22 842]/Annots 21 0 R/Parent 7 0
R/StructParents 0/Contents 29 0 R/Rotate 0/MediaBox[0 0 595.22
842]/Resources<</Font<</TT0                    30                    0
R>>/ProcSet[/PDF/Text]/ExtGState<</GS0 33 0 R>>>>/Type/Page>>
endobj

                                        ……

21 0 obj[22 0 R]
endobj

                                        ……

22 0 obj<</Rect[148.0 529.0 335.0 646.0]/Subtype/Widget/F 132/P 20
0    R/T(Signature2)/V    31    0    R/DA(/Helv    0    Tf    0
g)/FT/Sig/Type/Annot/MK<<>>/AP<</N 23 0 R>>>>
endobj

                                        ……

31                                                                  0
obj<</SubFilter/adbe.pkcs7.detached/Filter/Adobe.PPKLite/Contents<3
082033906092a864886f70d010702a082032a308203260201013…00>/M(D:200707
28135017+02’00’)/Name(Christoph P)/ByteRange[0 5580 7940 7442 ]
/Prop_Build<</Filter<</Name/Adobe.PPKLite/R 131101/Date(Dec 14 2004
02:27:38)>>/App<</TrustedMode     true/OS[/Win]/Name/Exchange-Pro/R
458752>>/PubSec<</R         131101/Date(Dec         14         2004
02:28:36)/NonEFontNoWarn true>>>>/Type/Sig>>
endobj
```

Code 25: Pfad zum Sign Objekt

Das Dokument wird durch das Objekt 31 signiert. Es wird in dem Bereich Content festgelegt, wie das Dokument zum Zeitpunkt des Signierens ausgesehen hat und auch wer das Dokument wann signiert hat.

Jede Änderung in dem Objekt 31 führt unweigerlich dazu, dass das Dokument beim Öffnen eine Fehlermeldung anzeigt.

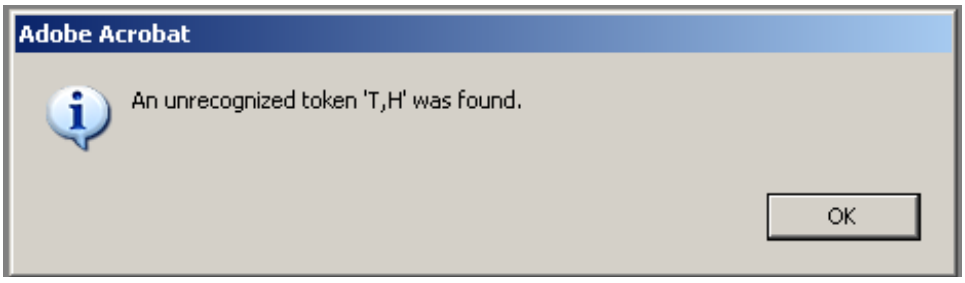


Abbildung 40: Fehlermeldung nach dem Verändern des Sign Objektes

Wird diese Fehlermeldung mit OK bestätigt, so öffnet sich das Dokument. Es ist jedoch zu sehen, dass das Unterschriftenfeld ungültig ist.

Abbildung 41: Ungültiges Dokumentenfeld

Wird nun versucht über das Dokumentenfeld die ursprüngliche Version des Dokuments über das Menü des Signaturfeldes anzuzeigen, so Funktioniert dies nicht. Es kommt zu folgender Fehlermaldung:

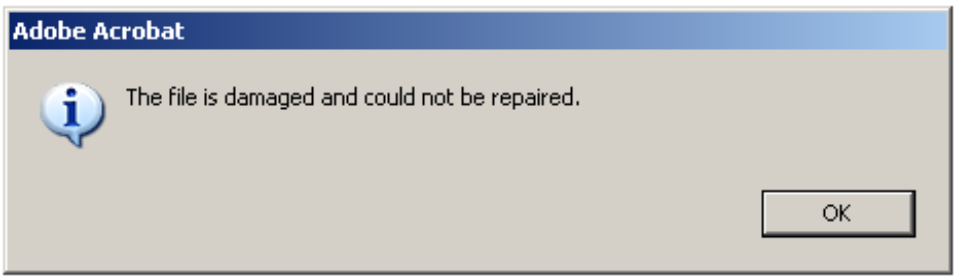


Abbildung 42: Vergleich von Signierter Dokumentenversion nach Änderung

Festgehalten kann werden, dass es nach einer Veränderung dieses Objektes zu einem unweigerlichen Erkennen einer Änderung in dem Dokument kommt. Der Benutzer kann zwar nicht mehr erkennen um welche Änderung es sich gehandelt hat, aber er weiß, dass sich das Dokument geändert hat.

10.3. Pdf 1.7 (Version 8)

In dem 2. Quartal 2007 wurde von Acrobat Adobe eine neue Version des Acrobat Studios mit der Version 8 was einer .pdf Spezifikation 1.7 entspricht herausgegeben. Wie auch alle Vorgängerversion ist diese Version mit allen Dokumenten die mit einer früheren .pdf Spezifikationsnummer erzeugt wurden kompatibel.

Die Version 8 benötigt auf dem Rechner wesentlich mehr Festplattenspeicher und es kommt aufgrund der erhöhten Komplexität bei nicht so starken Rechnern vermehrt zu abstürzen.

Weiters ist zu erwähnen, dass die .pdf Spezifikationen von Acrobat Adobe ab der Version 8 (.pdf Spezifikation 1.7) den ISO 32000 Normen entsprechen. [25]

Es hat sich auch der .pdf Code der hinter einem .pdf Dokument liegt bei einer Erstellung mit Acrobat Adobe 8 geändert. [24]

Obwohl die Dokumentenstruktur unverändert blieb, wurden deutliche Sicherheitsverbesserungen implementiert. Wie schon bekannt, ist in dem .pdf Code der Version 7 (pdf Spezifikation 1.6) sehr viel Text in Klartext geschrieben wenn das Dokument nicht gesichert ist. Es sind sowohl Erstellungs- und Updateinformationen enthalten, als auch Inhalte von Javascript. Weiters sind auch Informationen wie die Seitenzahl oder die Anzahl und die Arten von Formularfeldern auf einer Seite ersichtlich. Lediglich der „Content" einer Seite ist in einem „Stream" eingebettet, welcher mittels FlatEncode komprimiert ist.

Mit Acrobat Adobe 8 (pdf Spezifikation 1.7) sind nun fast alle Objekte in einem „Stream" eingebettet. Dies ist jedoch keine Änderung der .pdf Spezifikationen an sich, sondern es wird lediglich eine schon zuvor definierte Funktion angewendet. Laut den .pdf

Spezifikationen muss ein Objekt mit einem bestimmten Tag „Filter/FlateDecode“ und einem enthaltenen „Stream“ beim Auslesen dekomprimiert werden muss.

Da sich diese Anwendung bis jetzt nur auf den Inhalt einer Seite beschränkte, war der restliche Code lesbar.

Mit Version 8 ist jedoch nahezu jedes Objekt mit diesem Filter Tag und einem „Stream“ ausgestattet. Dadurch ist es nicht mehr möglich den .pdf Code in einem Editor zu lesen.

Dies führt dazu, dass sich die Sicherheit des Codes bezüglich Veränderungen im Code wesentlich verbessert hat.

Da jedoch wieder nur eine Komprimierung und keine Verschlüsselung vorgenommen wird, währe es weiterhin möglich, über das in Code 21 beschreiben Programm den Inhalt der einzelnen Objekte auszulesen.

```
Obj<</Length 74/C 84/Filter/FlateDecode/I 106/L 68/S 38>>
```

Code 26: Objekt mit Filter

Mit einem etwas komplexeren Programm wäre es natürlich auch möglich nach einer Ausgabe etwas zu verändern und dann wieder in den „Stream“ zurück zu schreiben. Hierbei ist jedoch zu sagen, dass nicht nur der „Stream“ neu geschrieben werden muss, sondern auch die Definition des Filters, da sich bei einer Änderung unweigerlich auch die Länge des „Streams“ ändert, welche in der Definition des Objektes unter „Length“ zu finden ist.

Weiters ist die .pdf Spezifikation von einer 8,5 MB Datei in Version 1.6 auf einer 36MB Datei gewachsen. Dies macht jegliche Suche in den .pdf Spezifikationen zu einem Geduldsspiel.

10.4. Zusammenfassung

Die Beschreibung/Spezifikation des .pdf Codes ist in Dokumenten die von Adobe zur Verfügung gestellt werden relativ ausführlich dargestellt. Hier werden Methoden und Funktionen des .pdf Codes beschrieben, die es möglich machen einen .pdf Viewer oder einen .pdf Drucker selbst zu schreiben. Jedoch ist zu sagen, dass in Sachen Sicherheit und Signierung einige Mängel in diesen Dokumenten auffallen.

Eine der grundlegenden Funktionen außer des Objektaufbaues in dem .pdf Code, ist das FlateDecode. Dabei handelt es sich um ein Komprimierungstechnik die sowohl für Text als auch für Bilder (im Speziellen im Format .png) verwendet werden kann. Durch die Komprimierung der Seiteninhalte mit diesem Verfahren ist es möglich, über das Objektmodel immer den Inhalt einer Seite, unabhängig davon, ob es sich um Text, Bilder oder Text mit Bildern handelt, sehr effizient zu speichern. Handelt es sich um ein nicht verschlüsseltes .pdf Dokument so kann dieser Code mittels eines Programms ausgelesen werden. Es ist also nicht so wie in anderen Programmen in denen der Inhalt schon im Klartext im Code gespeichert ist. Text in einem .pdf Dokument ist kryptisch, kann aber relativ einfach durch ein Programm extrahiert werden und in Klartext wiedergegeben werden.

Weiters kommt es durch diese Komprimierung zu einer geringeren Größe der .pdf Dokumente. Dieses Verfahren wurde bis Acrobat Adobe 7 nur auf den Seiteninhalt angewandt, und wird ab Version 8 auf fast alle Objekte in dem .pdf Code angewandt. Durch diese Änderung wir des extrem schwierig den .pdf Code zu verändern.

Eine weitere Sicherheit in einem .pdf Dokument ist die Vergabe von Passwörtern, welche entweder zum Öffnen eines Dokuments benötigt werden oder zum Erreichen des Rechtebereiches, wenn dort Sicherheitseinstellungen gemacht wurden.

Jedes Dokument das über ein Passwort verfügt, ist automatisch verschlüsselt, und jede Änderung an sinnvollen Stellen im Code führt entweder dazu, dass das Dokument als fehlerhaft gilt und nicht geöffnet werden kann, oder es sich nur ohne Inhalt der Seiten öffnen lässt.

Die Passwörter die vergeben wurden dienen dazu, den verschlüsselten Inhalt des Dokuments zu entschlüsseln und danach anzuzeigen. Da diese Passwörter einige Male einen MD5 Algorithmus durchlaufen bevor sie als Schlüssel dienen, ist es nicht möglich ein .pdf Dokument ohne den richtigen Schlüssel zu öffnen oder von dem verschlüsselten Text und dem Originaltext auf einen Schlüssel rückzuschließen.

Werden nicht nur Passwörter, sondern auch eine Public-Privat-Key System verwendet, so ist es sogar möglich jedem Benutzer eigene Rechte zu geben. Auch hier wird ein Schlüssel erstell, mit dem bestimmte Teile des Dokuments verschlüsselt werden.

Das Rechtesystem an sich bildet sich nur in einer Zahl ab die zwar zum Entschlüsseln des Textes verwendet werden muss, jedoch von anderen Anbietern von .pdf Viewer gegebenenfalls einfach ignoriert werden kann.

11. Alternative .pdf Viewer

In diesem Kapitel werden zwei alternative .pdf Reader auf deren Tauglichkeit und deren Funktionen überprüft. Bei diesen Programmen handelt es sich um den .pdf Reader von http://www.docu-track.com und http://www.foxitsoftware.com/. Beide Programme wurden so gewählt, dass sie nicht nur .pdf Dokumente anzeigen können, sondern auch Formularfelder unterstützen.

Programme wie GSView aus der GhostScript Produktserie können dies nicht und sind reine Darstellungsprogramme.

Bei beiden Programmen kommen .pdf Dokumente die mit der Version 7 (pdf code 1.6) erzeugt wurden zum Einsatz.

Die Änderungen in den .pdf Dokumenten werden immer in Acrobat Studio 7 erstellt und dann in den jeweiligen Readern geöffnet. Bei der Sicherung von Dokumenten wird dies immer auf Basis der Version Acrobat 7 durchgeführt.

11.1. Formularfelder ohne Javascript

Hierbei wurde ein Textfeld und Schaltflächen in die .pdf Datei aufgenommen, wobei eine Schaltfläche das rot hinterlegte Textfeld ausblendet, und die zweite Schaltfläche das Textfeld wieder einblendet.

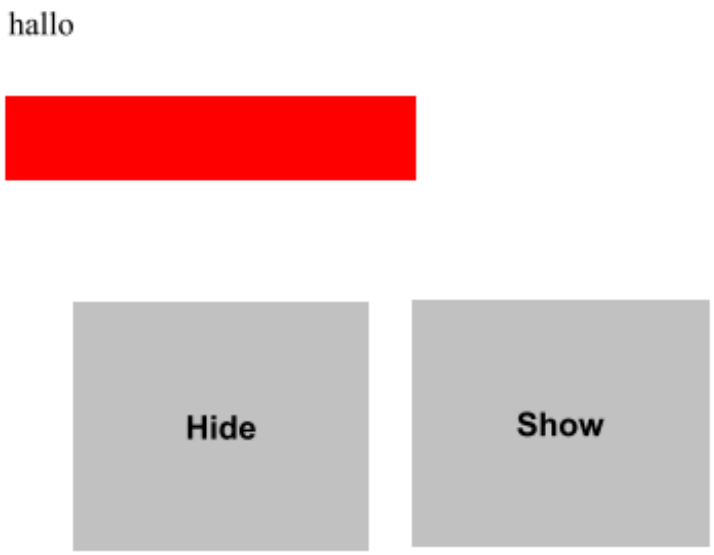


Abbildung 43: Test: Formularfeld ohne Javascript

Bei dem Start des Foxit Readers kam es wie in Abbildung 44 ersichtlich zu einem Dialog der nach dem Download einer Javascriptkomponente fragt. Diese Komponente wurde einmalig herunter geladen.

Abbildung 44: Komponentendownload bei dem Foxit Viewer

Bei beiden Programmen, dem Foxit Viewer und auch dem Docu Track Viewer, konnten die Formularfelder richtig dargestellt werden, und auch das Ein- und Ausblenden des Textfeldes funktionierte.

11.2. Formularfelder mit Javascript

Hier wurde nicht eine bereits vorhandene Funktion in dem Formularfeld benutzt um eine Aktion auszuführen, sondern mit der Aktion „Run Java Script" ein eigenes Verhalten erzeugt.

Wie in Abbildung 45 zu sehen soll beim Drücken der Schaltfläche eine Meldung mit dem Text Javascript geöffnet werden.

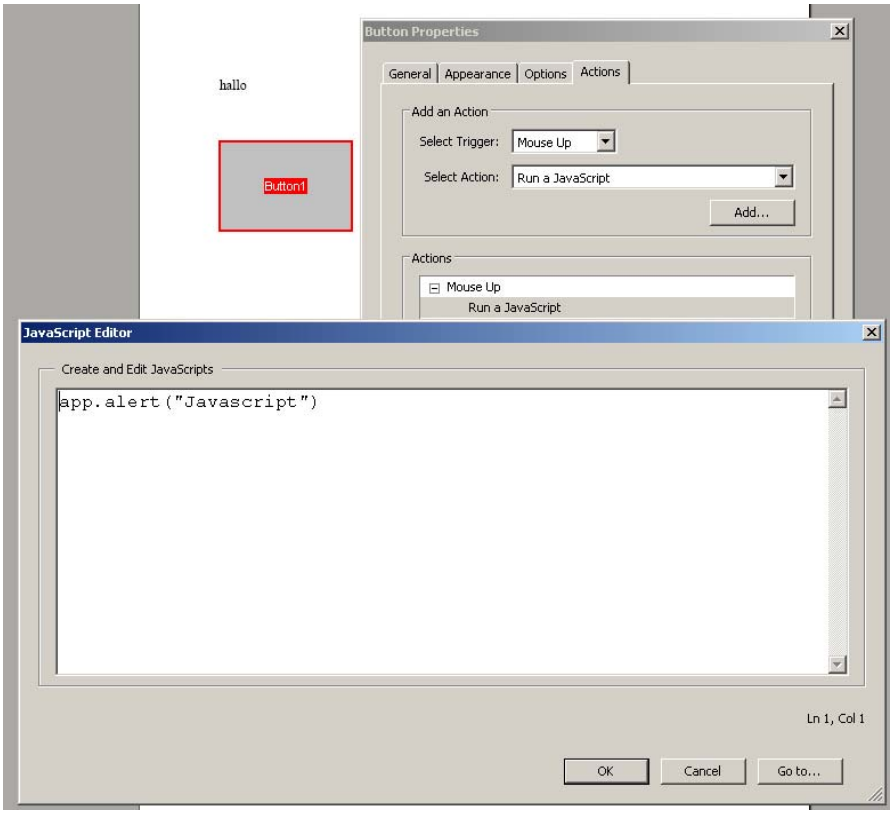


Abbildung 45: Test: Formularfeld mit Javascript

Die Funktionalität in den .pdf Viewern entspricht genau der in Acrobat. Es ist hier lediglich zu sagen, dass in dem „Alert" welcher geöffnet wird nicht Adobe Acrobat als Kopfzeile steht, sondern der jeweilig Programmname.

Außerdem hat die geöffnete Meldung in Adobe Acrobat und auch in dem Docu Track Viewer eine einheitliche Größe. In dem Foxit Viewer ist die Größe jedoch bei der ausgegebenen Meldung („Javascript") wesentlich kleiner. Bei einem längeren Text wird aber auch hier eine größere Meldung angezeigt. Jedoch ist die Meldung als Ganzes etwas anders skaliert als in Acorbat Adobe oder Docu Track Viewer, welche gleich skalieren.

11.3. Seiten- und Dokumentenvorgänge

Nun sollen auch Seitenvorgänge (Öffnen und Schließen einer Seite) sowie Dokumentenvorgänge (Dokument wird geschlossen, gespeichert, gedruckt und Dokument wurde gespeichert, gedruckt) ausgeführt werden.

Im folgenden Versuch wird auf jeden der Vorgänge ein eigenes Javascript, welches eine Meldung mit der jeweiligen Aktion ausgibt, gelegt.

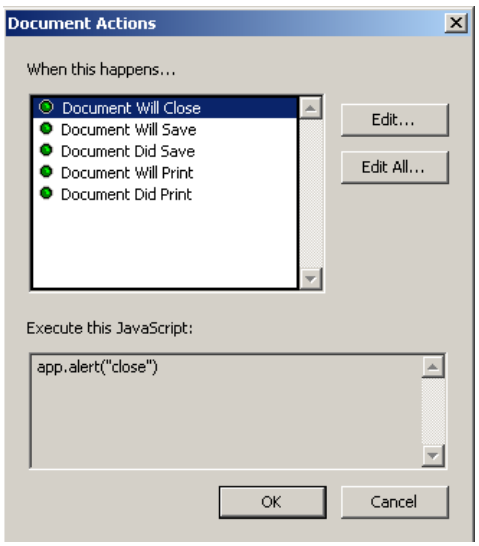


Abbildung 46: Test: Benutzte Dokumentenvorgänge

Dokumentenvorgänge verhalten sich in dem Docu Track Viewer gleich wie in dem Acrobat Adobe.

Leider werden die Vorgänge in dem Foxit Viewer zur Gänze ignoriert. Keiner der fünf Vorgänge wurde angezeigt, obwohl sicher ist, dass diese funktionieren, da sie auch in dem Docu Track Viewer angezeigt wurden. Weiters ist sichergestellt, das Javascript im Allgemeinen funktioniert, da auch der Test aus Abschnitt 11.2 ohne Probleme durchgeführt werden konnte.

Aufgrund der Ausschlussmethode kann also mit Sicherheit gesagt werden, dass der Foxit Viewer keine Dokumentenvorgänge unterstützt.

Der Seitenvorgang wurde auf die erste Seite des Dokuments angewandt und funktioniert in Acobat Adobe einwandfrei beim Öffnen des Dokuments.

Weder in dem Foxit Viewer, noch in dem Docu Track Viewer wird der Seitenvorgang beim Öffnen des .pdf Dokuments ausgeführt.

11.4. Rechteinschränkungen

In diesem Test wurden über Adobe Acrobat folgende Rechte gesetzt:

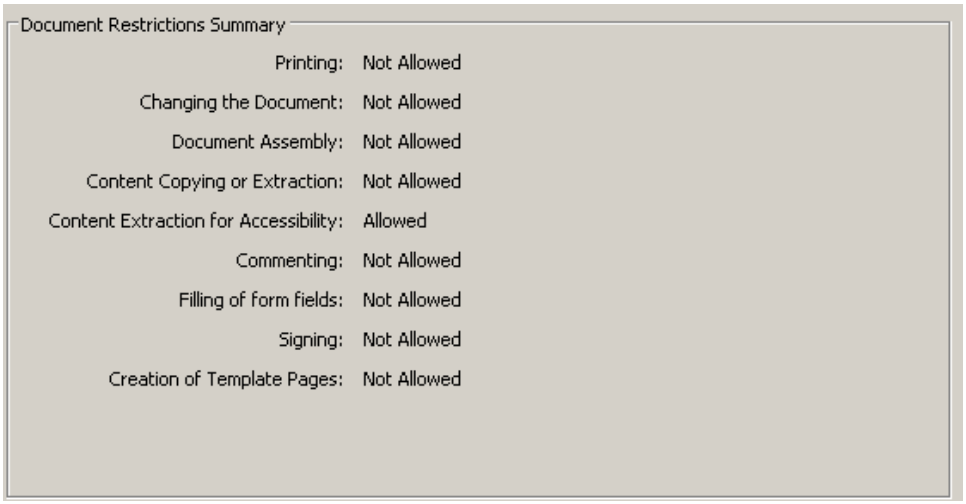


Abbildung 47: Test: Rechte-Einstellungen

Aufgrund dieser Einstellung sind Acobat Adobe die Möglichkeiten zum Drucken ausgegraut, und auch alle Möglichkeiten das Dokument zu ändern (wie das Hinzufügen von Kommentaren) sind nicht auswählbar. Es gibt auch aufgrund des .pdf Codes keine Möglichkeit die Rechte in dem Code zu ändern, und somit diese Einschränkungen zu umgehen.

Wird das Dokument in dem Docu Track Viewer geöffnet, so ist es sowohl erlaubt das Dokument zu Drucken als auch Kommentare einzufügen oder Text farbig zu hinterlegen. Nach einer Änderung, kann das Dokument auch weiterhin mit einem anderen Namen gespeichert werden.

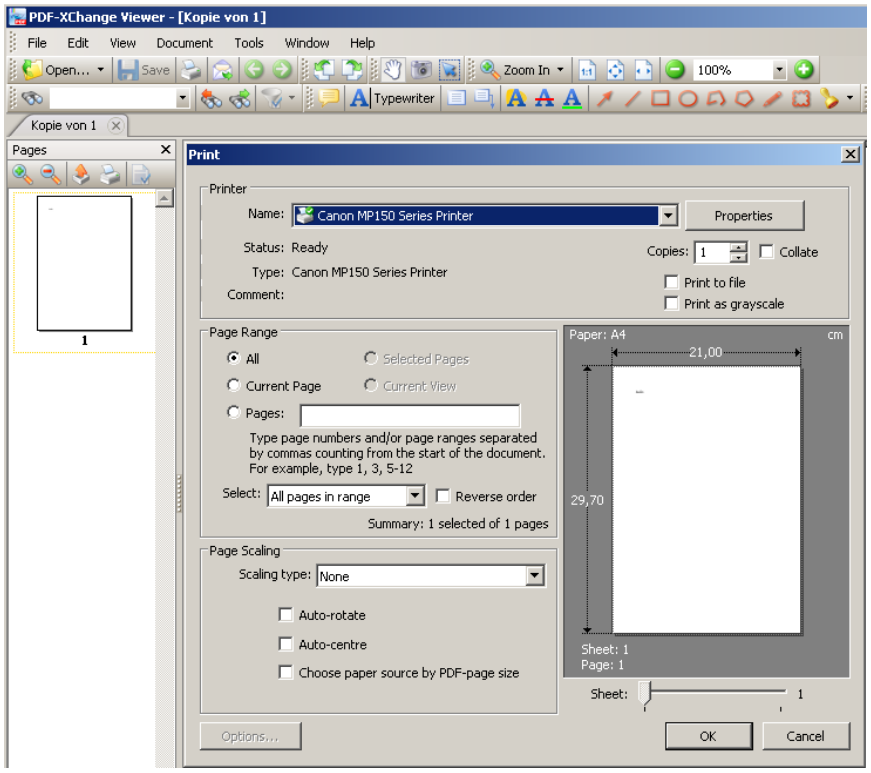


Abbildung 48: Test: Drucken trotz Drucksperre (Docu Track Viewer)

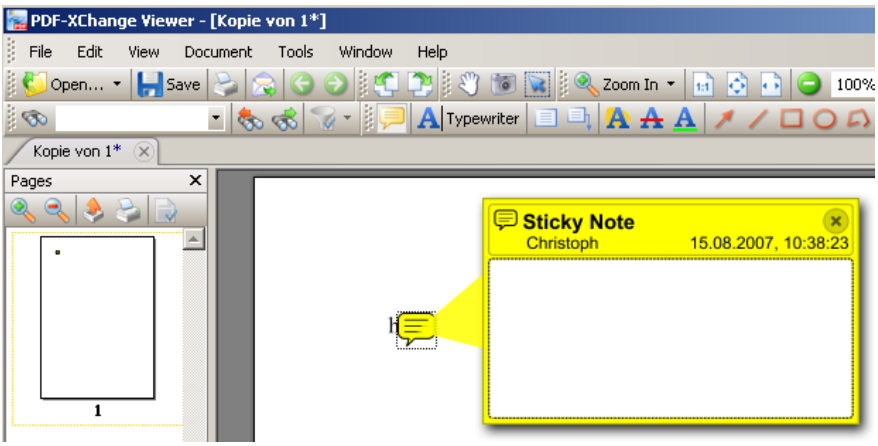


Abbildung 49: Test: Textbearbeitung trotz Sperre (Docu Track Viewer)

In dem Foxit Viewer ist es nicht erlaubt das Dokument zu drucken. Trotzdem kann man ohne Probleme Kommentare einfügen, oder Text farbig hinterlegen.

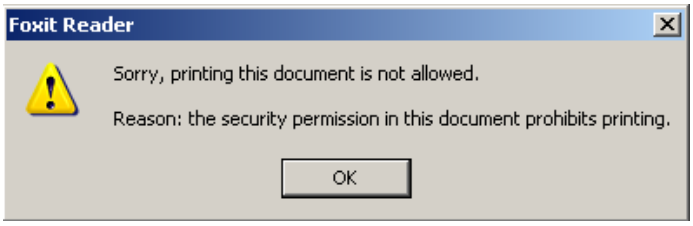


Abbildung 50: Test: Drucksperre (Foxit Viewer)

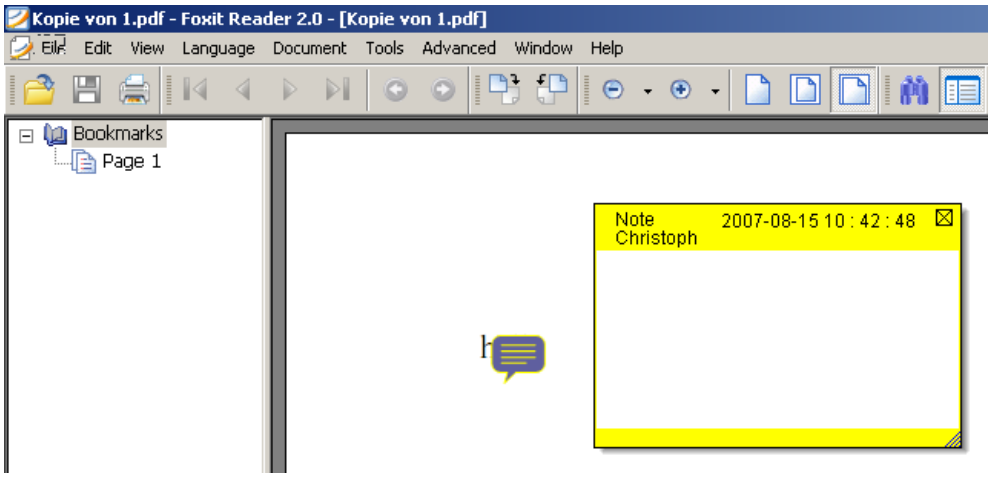


Abbildung 51 Test: Textbearbeitung trotz Sperre (Foxit Viewer)

Hierbei ist noch einmal zu erwähnen, dass sich die Rechte eines Dokuments lediglich in einer 4-stelligen Zahl in dem .pdf Code widerspiegeln, welche zwar zum Entschlüsseln des Dokumenteninhalts verwendet werden muss, jedoch keinerlei weitere Funktion hat. Sie kann also von dem ausführenden Programm ignoriert werden.

Anscheinend werden die Rechte auf ein Dokument von dem Docu Track Viewer ignoriert und somit ist sowohl der Druck als auch die Veränderung des Dokuments möglich.

Was hier jedoch eher unangenehm auffällt ist, dass der Foxit Viewer die Rechte nicht ignoriert, da man nicht drucken kann. Die Veränderung des Dokuments ist aber trotz eingeschränkter Rechte möglich ist. Es werden hier also manche Rechte berücksichtigt, und manche nicht.

11.5. Öffnen-Passwort

In diesem Test wird das Dokument mit einem Passwort zum Öffnen des Dokuments versehen.

Sowohl der Docu Track Viewer als auch der Foxit Viewer können das Dokument nicht ohne das richtige Passwort öffnen.

Abbildung 52: Test: Passwortabfrage (Docu Track Viewer)

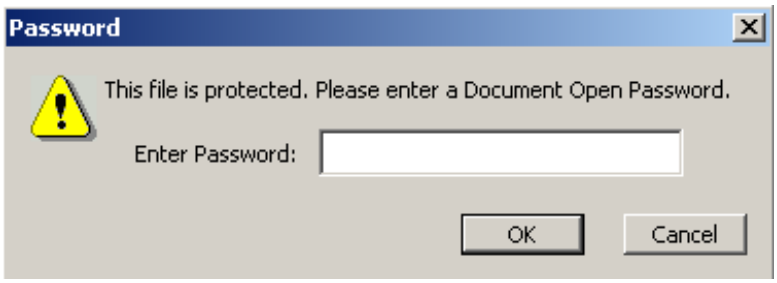


Abbildung 53 Test: Passwortabfrage (Foxit Viewer)

11.6. Signierung

In diesem Test soll überprüft werden, ob in dem Foxit Viewer und in dem Docu Track Viewer auch Signierungen richtig angezeigt werden.

Dazu wird ein Dokument in dem Adobe Acrobat signiert und in den beiden Viewern geöffnet. Weiters wird das Dokument in Adobe Acrobat verändert, und erneut in dem Foxit Viewer und in dem Docu Track Viewer geöffnet.

Es wird überprüft, ob die Signierung in den beiden Viewer vorhanden ist und ob auch die Gültigkeit der Signierung deutlich erkennbar ist.

Beim Öffnen des gültig signierten Dokuments in dem Foxit Viewer ist ersichtlich, dass das Signierungsfeld zwar angezeigt wird, aber es sich hierbei lediglich um ein Bild handelt. Es ist nicht möglich über das Kontextmenü oder über einen anderen Pfad die Gültigkeit der Signierung abzufragen. Weiters ist auch das Zeichen welches den Staus der Signierung in dem Dokument angibt nicht ersichtlich.

Auch in dem Docu Track Viewer ist weder eine Möglichkeit der Abfrage für die Signierung gegeben, noch wird das Bildchen für den Status der Signierung angezeigt.

Auch wenn die Signierung ungültig ist, ist das Verhalten in den beiden Viewern gleich wie bei einer gültigen Signierung.

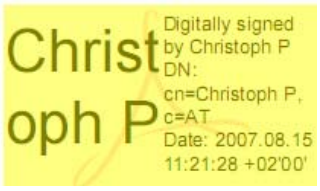
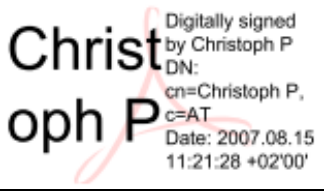
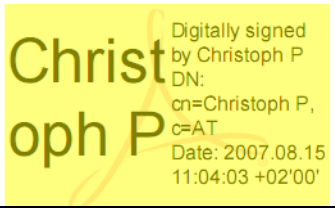
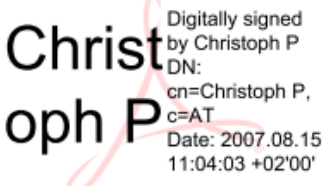
Signierung	Acorbat Adobe	Foxit Viewer	Docu Track Viewer
Gültig			
Ungültig			

Tabelle 14: Test: Signierung

11.7. Zusammenfassung

Generell eröffnen die untersuchten .pdf Viewer die Möglichkeit ein .pdf Dokument anzusehen und können verschlüsselte Dokumenten lesen. Auch die Sicherheit beim Öffnen eines .pdf Dokuments ist durch diese Programme gewährleistet. Befinden sich in einem Dokument Javascriptvorgänge auf Formularfeldern so werden diese (unter Umständen nach dem Download einer zusätzlichen Komponenten) richtig ausgeführt.

Große Schwächen zeigen die Viewer jedoch bei der Signierung von Dokumenten, wo nicht klar ist, ob die Signierungen gültig sind oder nicht. Obwohl die Signierungsfelder dargestellt werden. So kann es sehr schnell zu dem Eindruck kommen, das Dokument signiert und unverändert ist, obwohl dies nicht zutrifft.

Auch Seitenvorgänge wurden nicht angezeigt und Dokumentenvorgänge von nur einem der beiden Viewer dargestellt.

Bei der Einhaltung der Rechte zeigen die Viewer ebenfalls gravierende Schwächen indem Rechte welche die Änderung verbieten nicht umgesetzt sind, und auch Rechte die das Drucken verhindern nur von einem der beiden Programme berücksichtigt sind.

Allgemein ist zu sagen, dass es sich bei dem Acrobat Reader um eine freies Programm zum ansehen von .pdf Dokumenten handelt, welches sowohl mit Signierungen umgehen kann, als auch mit Seiten- und Dokumentenvorgängen. Um die Sicherheit und Funktion zu wahren ist es von Vorteil dieses Programm zu benutzen.

Möchte man die Sicherheit noch erhöhen, so ist es in dem Acrobat Reader auch denkbar, das Javascript zu deaktivieren, was bei keinem der Viewer erreicht werden kann.

Ein Vorteil für den Benutzer und Nachteil für den Ersteller ist sicherlich die Tatsache, dass einer der beiden Viewer die Drucksperre übergeht. Somit ist es nicht möglich ein Dokument zu Sperren. Es kann weiterhin in dem entsprechenden Viewer geöffnet werden, und einfach ausgedruckt werden.

Für den normalen Gebrauch empfiehlt sich die Verwendung des originalen Produkts (Acrobat Reader). Will man jedoch Rechte übergehen, ist die Verwendung eines der hier vorgestellten Viewer sicherlich eine gute Alternative.

12. Konklusion

In den letzten Jahren wurde mit dem Versenden von Schriftstücken das .pdf Format zu einem unweigerlichen Begleiter. Egal ob für Abrechnungen von Dienstleistern, Bestätigungen von Onlinereisbuchungen oder für Formularen im Internet werden .pdf Dokumente branchenübergreifend eingesetzt.

Kalre Vorteile von .pdf Dokumenten sind, dass sie nicht so einfach wie etwa Microsoft Word Dokumente verändert werden können. Denn um ein .pdf Dokument zu verändern, braucht man nicht nur einen der im Internet frei zugänglichen .pdf Viewer, sondern muss sich ein eigenes Programm kaufen welches die Bearbeitung dieser Dokumente unterstützt. Auch die Erstellung einer .pdf Datei kann mit im Internet frei verfügbaren .pdf Druckern sehr schnell und ohne finanziellen Aufwand realisiert werden.

Somit ergibt sich eine kostenfreie Möglichkeit .pdf Dokumente zu erstellen und diese zu versenden.Man auch sicher sein, dass sie der Empfänger mit einem kostenlosen Programm auch ansehen kann.

Doch aufgrund des sorglosen Umgangs und des entstandenen Sicherheitsgefühls, entstehen riesige Sicherheitslücken in der Verwendung von .pdf Dokumenten. So ist es nicht allgemein bekannt, dass .pdf Dokumente auch Javascript enthalten, oder sich zur Laufzeit verändern können. Die Möglichkeit Javascript in .pdf Dokument einzubauen ist aus der Anforderung entstanden .pdf Dokumente als Onlineformulare zu verwenden. Die Javascripte sollen eigentlich das Ausfüllen von Formularen erleichtern oder die Fehleranfälligkeit verringern.

Ein Kernelement von .pdf Dokumenten sind Formularfelder. Formularfelder sind Felder wie Schaltflächen oder auch Texteingabefelder, welche nicht nur formatiert werden können, sondern auch bestimmte Auslöser haben an die man Vorgänge hängen kann.

Auslöser sind Aktionen wie „Maustaste drücken" oder „Maustaste loslassen" wenn sich die Maus über dem Formularfeld befindet. An diese Auslöser kann man nun Vorgänge hängen wie „Menübefehl ausführen", „Felder verstecken" oder „Javascript ausführen". Bei allen Vorgängen außer „Javascript ausführen" muss der Benutzer über keine besonderen Kenntnisse verfügen, da sich die Aktionen in einer jeweils eigenen Oberfläche durch „Klicken" konfigurieren lassen.

Mit diesen Aktionen ist es einfach möglich mit einem Klick auf ein Feld in dem .pdf Dokument etwas zu verstecken und ein anders Feld mit anderem Inhalt zu zeigen, ohne das der Benutzer etwas merkt.

Mit den vordefinierten Vorgängen kann man jedoch nicht auf einen anderen Feldwert reagieren. Um dies zu realisieren muss als Vorgang „Javascript ausführen" gewählt werden, und in dem Javascript selbst der entsprechende Code implementiert werden. Hier ist es dann möglich auch komplexere oder von anderen Feldwerten abhängige Aktionen auszuführen.

Ein weiterer Bereich in dem man mit Javascript arbeiten muss sind die so genannten Seiten- und Dokumentenvorgänge. Seitenvorgänge beziehen sich auf das Öffnen und Schließen einer Seite. Dokumentenvorgänge sind Auslöser vor und nach dem Speicher, vor und nach dem Drucken, sowie beim Schließen des Dokuments. Mit den Dokumentenvorgängen welche vor und nach einer Aktion ausgeführt werden ist es sogar möglich, einen Inhalt zu speichern oder zu drucken den der Benutzer nie sieht. Dazu ist es lediglich notwendig sich den Wert eines Feldes in ein unsichtbares Formularfeld zu speichern, den sichtbaren Wert vor der Aktion zu verändern und nach der Aktion den alten Wert aus dem anderen Formularfeld wieder in das sichtbare Feld zu schreiben. Mit solchen Veränderungen ist eine Reihe an Betrugsszenarien gut vorstellbar.

Eine weitere aber nicht so einfache Möglichkeit ein .pdf Dokument zu verändern, ist der Eingriff in den .pdf Code selbst. Dies kann dann erforderlich sein, wenn man ein .pdf Dokument zwar verändern möchte, aber nicht will, dass in dem Dokument aufscheint, dass das Dokument verändert wurde.

In einem .pdf Dokument ist es ersichtlich wer wann eine Änderung an dem Dokument vollzogen hat. Dieses System kann nur durch den Eingriff in dem .pdf Code selbst übergangen werden. Um jedoch in dem Code Änderungen durchzuführen, ist es notwendig sich mit dem Code auseinanderzusetzen um den Objektaufbau des .pdf Codes zu verstehen, und auch an den richtigen Stellen die richtigen Schlüssel für einen Auslöser einzufügen. Ist dies geschehen muss der Schlüssel nur noch mit einer nicht vorhanden Objektnummer versehen werden, und das neue Objekt eingefügt und mit Javascript ausprogrammiert werden.

Um diese Möglichkeit zu unterbinden, muss von dem Ersteller entweder ein Recht auf das Dokument gesetzt werden, oder das Dokument mit einem Passwort zum Öffnen versehen werden. In beiden Fällen wird der Code nicht mehr in Klartext gespeichert, sondern es kommt zu einer Verschlüsselung des Codes indem auf das Passwort einige Male ein MD5 Algorithmus angewandt wird. Die entstandene Zeichenfolge wird als Schlüssel für die Verschlüsselung verwendet. Durch die Verwendung des MD5 Algorithmus ist auch ein Rückschließen auf das Passwort nahezu unmöglich. Wird dennoch versucht in dem verschlüsselten Code etwas zu ändern, kommt es unweigerlich zu Problemen beim Öffnen des Dokuments. Entweder das Dokument lässt sich nicht mehr öffnen oder es wird leer geöffnet.

Weiters stellt Acrobat auch eine Methode für eine Public-Private-Key Verschlüsselung zur Verfügung, bei der jeder Benutzer eigene Rechte in dem .pdf Dokument hat. Aber auch bei dieser Methode wird der Code verschlüsselt und eine Änderung des Codes ist nicht möglich.

Um ein Dokument nicht vollständig sichern zu müssen, jedoch von einem unveränderten Inhalt ausgehen zu können, gibt es auch die Möglichkeit ein Dokument zu Signieren. Dabei führt auch jede Änderung in dem .pdf Code dazu, dass das Dokument nicht mehr geöffnet oder leer geöffnet wird. Zusätzlich hat man bei Acrobat Adobe die Möglichkeit zu sehen, ob sich der Inhalt des Dokuments geändert hat. Es ist auch möglich eine unterschrieben Version mit der vorhandenen Version zu vergleichen und alle Änderung in dem Dokument zu sehen. Eine Unterschrift in einem Dokument kann gültig, gültig aber mit Veränderungen oder ungültig sein.

Ein weiter Vorteil von .pdf Dokumenten ist eine relativ geringe Dokumentengröße, welche dadurch entsteht, dass der Inhalt einer Seite mit ein Deflate Algorithmus komprimiert wird. Diese Art der Komprimierung wird auch bei Bildern im Format .png verwendet und ist sehr gut geeignet um grafischen Inhalt und Text zu komprimieren.

Wie bereits erwähnt ist der .pdf Viewer von Acrobat „Acrobat Reader" nicht das einzige Programm um .pdf Dokumenten darzustellen. Es gibt im Internet einer Vielzahl von Produkten mit denen man .pdf Dokumente ansehen kann. Hier ist aber zu sagen, dass es angefangen von Produkten die keine Formularfelder anzeigen, bis hin zu Produkten die sogar Javascript beherrschen, alles gibt. Entscheidet man sich jedoch für ein solches Produkt, muss man auf jeden Fall davon ausgehen, dass nicht alle Funktionen die auch der Acrobat Reader unterstützt von diesen Produkten unterstützt werden.

Für den Benutzer positiv ist allerdings zu sagen, dass manche dieser Produkte über die eingestellten Rechte hinwegsetzen, und somit auch für gesperrte Dokumente das Einfügen von Kommentaren oder auch das Drucken des Dokuments möglich ist.

Abschließend ist festzuhalten, dass es für den Ersteller eines Dokuments immer Möglichkeiten gibt, Codeteile einzuarbeiten, die das Dokument verändern. Ob dies nun zum Positiven (Hilfestellung) oder zum Negativen (absichtlich falsche Berechnungen) ist kann nicht gesagt werden.

Ist ein Dokument aber geschützt (in irgendeiner Form verschlüsselt) ist es für dritte nicht im Code veränderbar, und wenn entsprechende Rechte gesetzt sind auch im Programm nicht veränderbar.

13. Referenzen

[1]	http://de.wikipedia.org/wiki/PDF
[2]	http://de.selfhtml.org/
[3]	http://www.w3schools.com/js/default.asp
[4]	http://www.adobe.com/devnet/acrobat/javascript.html
[5]	http://de.wikipedia.org/wiki/Advanced_Encryption_Standard
[6]	http://de.wikipedia.org/wiki/Symmetrisches_Kryptosystem
[7]	http://de.wikipedia.org/wiki/Blockchiffre
[8]	http://www.kuno-kohn.de/crypto/crypto/rc4.htm
[9]	http://www.itwissen.info/definition/lexikon//__stream%20cipher_stromchiffre.html
[10]	http://www.zlib.net/feldspar.html
[11]	http://de.wikipedia.org/wiki/Message-Digest_Algorithm_5
[12]	http://www.kuno-kohn.de/crypto/crypto/rc4.htm
[13]	http://www.adobe.com/devnet/pdf/pdfs/PDFReference16.pdf
[14]	http://www.zdnet.de/downloads/prg/0/6/de10313206-wc.html (foxit pdf reader)
[15]	http://www.docu-track.com/home/prod_user/pdfx_viewer/
[16]	RFC1321: The MD5 Message-Digest Algorithm; Autor:R. Rivest; Datum: April 1992; Seite: 1
[17]	Proceedings of the fifteenth ACM conference on Hypertext and hypermedia; Autor: James C. King; Datum: 2004; Seite: 97
[18]	Cryptography and data security; Autor: Dorothy Elizabeth Robling Denning Datum: 1982
[19]	JavaScript instrumentation for browser security; Autor: Dachuan Yu, Ajay Chander, Nayeem Islam, Igor Serikov; Datum 2007; Seite: 237
[20]	Hardware Engines for Bus Encryption: A Survey of Existing Techniques; Autor: R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, C. Anguille, M. Bardouillet, C. Buatois, J. B. Rigaud; Datum: 2005; Seite: 2
[21]	DEFLATE Compressed Data Format Specification; Autor: L. Peter Deutsch; Datum: 1996
[22]	Das Dateiformat PDF im Web – eine statistische Erhebung; Autor: Philipp Mayr; Datum: Sommer 2005
[23]	PDF Reference – Adobe® Portable Document Format Version 1.6; Autor: Adobe Systems Incorporated; Date: November 2004
[24]	Errata for the PDF Reference, sixth edition, version 1.7; Autor: Adobe Systems Incorporated; Date: November 2006
[25]	About the ISO Draft of the PDF 1.7 Reference; Autor: Adobe Systems Incorporated; Date: 4. Juni 2007
[26]	On the Validity of Digital Signatures; Autor: Jianying Zhou, Robert Deng; Date: 2000; Seite:29
[27]	Adobe Acrobat 7.0.5 – Acrobat JavaScript Scripting Guide; Autor: Adobe Systems Incorporated; Date: 27.September 2005
[28]	JavaScript; Autor: Wenz, Christian; Datum:2005

[29]	Adobe Reader – Benutzerhandbuch 7.0; Autor: Adobe Systems Incorporated; Date: 2005
[30]	A methodology to implement block ciphers in reconfigurable hardware and its application to fast and compact AES RIJNDAEL; Autor: François-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, Jean-Didier Legat; Datum:2003
[31]	Sicherheit aktuell verwendeter Stromchiffren; Autor: Oliver Stutzke, Bernhard Löhlein; Datum: 23.Mai 2000
[32]	Public Key Cryptography - Applications Algorithms and Mathematical Explanations; Autor: Tata Elxsi Ltd, India; Seite: 1

14. Abbildungs- und Tabellenverzeichnis

14.1. Abbildungsverzeichnis

14.2. Codeverzeichnis

14.3. Tabellenverzeichnis