PDF automatisiert testen

PDFUnit-Perl

Carsten Siedentop

Inhaltsverzeichnis

Vorwort

Aktuelle Testsituation in Projekten

Telefonrechnungen, Versicherungspolicen, amtliche Bescheide, Verträge jeglicher Art werden heute als PDF-Dokumente elektronisch zugestellt. Ihre Erstellung erfolgt in vielen Programmiersprachen mit zahlreichen Bibliotheken. Je nach Komplexität der zu erstellenden Dokumente ist diese Programmierung nicht einfach und enthält wie jede Software auch Fehler, die eventuell zu fehlerhaften PDF Documenten führen. Deshalb sollte geprüft werden:

- Steht in einem bestimmten Bereich einer Seite der erwartete Text?
- Stimmt der Barcode (QR-Code) auf dem Dokument mit dem erwarteten Inhalt überein?
- Stimmt das Layout mit der Vorgabe überein?
- Stimmen die Werte der eingebetteten ZUGFeRD-Daten mit den erwarteten Daten überein?
- Stimmen die Werte der eingebetteten ZUGFeRD-Daten mit den sichtbaren Daten überein?
- Entspricht ein Dokument den Regeln von DIN 5008?
- Ist das PDF signiert? Wann und von wem?

Es sollte Entwickler, Projekt- und Unternehmensverantwortliche erschrecken, dass es bisher kaum Möglichkeiten gibt, PDF-Dokumente **automatisiert** zu testen. Und selbst diese Möglichkeiten werden im Projektalltag nicht genutzt. Manuelles Testen ist leider weit verbreitet. Das ist teuer und fehleranfällig.

Egal, ob PDF-Dokumente mit einem mächtigen Design-Werkzeug, mit MS-Word/LibreOffice oder eigenen Programmen erstellt werden oder ob sie aus einem XSL-FO Workflow herausfallen, jedes PDF-Dokument kann mit PDFUnit getestet werden.

Intuitive Schnittstelle

Die Schnittstelle von PDFUnit-Perl ist ein Wrapper um die Schnittstelle von PDFUnit-Java. Die Funktionsnamen lehnen sich eng an die Umgangssprache an und unterstützen damit gewohnte Denkstrukturen. Dadurch entsteht Code, der auch langfristig noch leicht zu verstehen ist.

Wie einfach die Schnittstelle konzipiert ist, zeigt das folgende Beispiel:

```
my $filenameTest   = "$PATH/master/compareToMaster.pdf";
my $filenameMaster = "$PATH/master/compareToMaster_protected.pdf";
my $userPassword   = 'user-password';
lives_ok(
  AssertThat
    ->document($filenameTest)
    ->and($filenameMaster, $userPassword)
    ->haveSameText(ON_FIRST_PAGE)
  , "compareTextAgainstEncryptedMaster_OnSinglePage"
);
```

Ein Test-Entwickler muss weder Kenntnisse über die Struktur von PDF haben, noch etwas über die fachliche Enstehungsgeschichte des PDF-Dokumentes wissen, um erfolgreiche Tests zu schreiben.

Zeit, anzufangen

Spielen Sie nicht weiter Lotto bei der Erstellung Ihrer PDF-Dokumente. Überprüfen Sie das Ergebnis Ihrer PDF-Erstellung durch automatisierte Tests.

Kapitel 1. Über diese Dokumentation

Wer sollte sie lesen

Die vorliegende Dokumentation richtet sich in erster Linie an Perl-Programmierer, deren Aufgabe es ist, dafür zu sorgen, dass technisch erzeugte PDF-Dokumente „richtig“ sind.

Es wird davon ausgegangen, dass Sie Grundkenntnisse in der Perl-Programmierung besitzen. Ebenfalls ist ein Grundverständnis über Testautomatisierung hilfreich, aber keine Voraussetzung.

Code-Beispiele

Die in den nachfolgenden Kapiteln abgebildeten Code-Beispiele verwenden das Modul `Test::More`. `PDF::PDFUnit` (PDFUnit-Perl) kann aber auch in Verbindung mit jedem anderen Testmodul verwendet werden. Im Modul `PDF::PDFUnit` sind einige Beispiele enthalten.

Referenzen zur Java-Dokumentation

Die hier vorliegende Dokumentation enthält (nur) so viele Beispiele, dass die Benutzung von PDFUnit-Perl klar wird. Weil sich die Syntax und die Bedeutung einer Testmethode nicht von der Java-API unterscheidet, wird in dieser Dokumentation nicht jede Testfunktion beschrieben. Ziehen Sie die separate Dokumentation von PDFUnit-Java hinzu, um jede Funktion detailiert kennenzulernen. Sie ist online (http://www.pdfunit.com/de/documentation/java/index.html) verfügbar.

API für PDFUnit-Java

Die Javadoc-Dokumentation der API ist online verfügbar: http://www.pdfunit.com/api/java-doc/index.html.

Andere Programmiersprachen

PDFUnit gibt es für Perl, Java .NET, und als XML-Implementierung. Für jede Sprache existiert eine eigene Dokumentation.

Wenn es Probleme gibt

Haben Sie Schwierigkeiten, ein PDF zu testen? Recherchieren Sie zuerst im Internet, vielleicht ist dort ein ähnliches Problem schon beschrieben, eventuell mit einer Lösung. Sie können die Problembeschreibung auch per Mail an info[at]pdfunit.com schicken.

Neue Testfunktionen gewünscht?

Hätten Sie gerne neue Testfunktionen, wenden Sie sich per Mail an info[at]pdfunit.com. Das Produkt befindet sich permanent in der Weiterentwicklung, die Sie durch Ihre Wünsche gerne beeinflussen dürfen.

Verantwortlichkeit

Manche Code-Beispiele in diesem Buch verwenden PDF-Dokumente aus dem Internet. Aus rechtlichen Gründen stelle ich klar, dass ich mich von den Inhalten distanziere, zumal ich sie z.B. für die chinesischen Dokumente gar nicht beurteilen kann. Aufgrund ihrer Eigenschaften unterstützen diese Dokumente Tests, für die ich keine eigenen Testdokumente erstellen konnte - z.B. für chinesische Texte.

Danksagung

Axel Miesen hat die Perl-Schnittstelle für PDFUnit entwickelt und in dieser Zeit viele Fragen zur Java-Version gestellt, die sich auf die Entwicklung von PDFUnit-Java vorteilhaft auswirkten. Herzlichen Dank, Axel.

Bei meinem Kollegen John Boyd-Rainey möchte ich mich für die kritischen Fragen zur Dokumentation bedanken. Seine Anmerkungen haben mich dazu bewogen, manchen Sachverhalt anders zu formulieren. John hat außerdem die englische Fassung dieser Dokumentation Korrektur gelesen. Die Menge der aufgedeckten Komma- und anderer Fehler muss eine Tortur für ihn gewesen sein. Herzlichen Dank, John, für Deine Ausdauer und Gründlichkeit. Die Verantwortung für noch vorhandene Fehler liegt natürlich ausschließlich bei mir.

Herstellung dieser Dokumentation

Die vorliegende Dokumentation wurde mit DocBook-XML erstellt. Die PDF- und die HTML-Version stammen aus einer einzigen Textquelle und sind somit inhaltlich identisch. In beiden Zielformaten ist das Layout noch verbesserungswürdig, wie beispielsweise die Seitenumbrüche im PDF-Format. Die Verbesserung des Layouts steht schon auf der Aufgabenliste, jedoch gibt es noch andere Aufgaben mit höherer Priorität.

Feedback

Jegliche Art von Feedback ist willkommen, schreiben Sie einfach an info[at]pdfunit.com.

Kapitel 2. Quickstart

Quickstart

Angenommen, Sie haben ein Projekt, das PDF-Dokumente erzeugt und möchten sicherstellen, dass die beteiligten Programme das tun, was sie sollen. Und angenommen, ein Test-Dokument soll genau eine Seite umfassen sowie die Grußformel „Vielen Dank für die Nutzung unserer Serviceleistungen“ und die Rechnungssumme von „30,34 Euro“ enthalten. Dann testen Sie diese Anforderungen so:

```
use strict;
use utf8;
use warnings;

use PDF::PDFUnit;
use Test::Exception;
use Test::More;

my $resources_dir = ...;   # let it point to the base folder of your test data
my $pdfUnderTest = "$resources_dir/quickstart/quickstartDemo_en.pdf";

#
# Testcase 'has one page'
#
lives_ok {
  AssertThat
    ->document($pdfUnderTest)
    ->hasNumberOfPages(1)
  ;
} "has one page";

lives_ok {
  my $expectedGreeting = "Vielen Dank für die Nutzung unserer Serviceleistungen";

  AssertThat->document($pdfUnderTest)
            ->restrictedTo(LAST_PAGE)
            ->hasText()
            ->containing($expectedGreeting)
  ;
} "has greeting";

#
# Testcase 'has expected charge'
#
lives_ok {
  my $upperLeftX  =  172;  # in millimeter
  my $upperLeftY  =  178;
  my $width       =   20;
  my $height      =    9;
  my $regionCharge = PageRegion->new($upperLeftX, $upperLeftY, $width, $height);

  AssertThat->document($pdfUnderTest)
            ->restrictedTo(LAST_PAGE)
            ->restrictedTo($regionCharge)
            ->hasText()
            ->containing('29,89')   # Let's see an error message. Correct value: 30,34
  ;
} "has expected charge";

diag $@->getMessage() if $@;
done_testing();
```

Die Anweisung `diag $@->getMessage() if $@` sorgt dafür, dass Fehlermeldungen von PDFUnit-Java angezeigt werden. Für den provozierten Fehler im letzten Test sieht die Fehlermeldung folgendermaßen aus:

```
# Failed test 'has expected charge'
# at C:/.../examples_quickstart-de.t line 67.
# died: main::com::pdfunit::errors::PDFUnitValidationException=HASH(0x4055938)
# Page(s) [1] of 'C:\...\quickstartDemo_de.pdf'
# not containing the expected text: '29,89 Euro'. Found: '30,34 Euro'.
```

So einfach geht's. Die folgenden Kapitel zeigen den Funktionsumfang, die Installation und typische Beispiele für die Verwendung von PDF::PDFUnit.

Kapitel 3. Funktionsumfang

3.1. Überblick

Syntaktischer Einstieg

Alle Tests benötigen die use-Anweisung `use PDF::PDFUnit`. Funktionen aus `Test::Exception` und `Test::More` werden in den Beispielen benutzt, gehören jedoch nicht zu PDFUnit.

```
use PDF::PDFUnit;
use Test::Exception;
use Test::More;
```

Tests auf ein **einzelnes PDF-Dokument** beginnen mit der Benennung der zu testenden Datei über die Methode `AssertThat->document(..)`. Von dort aus verzweigen die Tests in unterschiedliche Testbereiche, wie z.B. Inhalt, Schriften, Layout etc.:

Unter Verkürzung des Listungs um die notwendigen `use`-Anweisungen wird die Art der API im folgenden Beispiel deutlich:

```
my $resources_dir = ...;              # the base folder of your test data
my $pdfUnderTest = "$resources_dir/doc-under-test.pdf";    ❶

lives_ok {                            # function from Test::Exception
  my $expectedText = "John Doe";      #
                                      #
  AssertThat->document($pdfUnderTest) # document under test
            ->restrictedTo($page2)    # page selection
            ->hasText()               # test scope
            ->...                     # further validation methods
  ;                                   #
} "name of the test";                 # name of the test


#
# Comparing a single document with a reference:
#
lives_ok {                            #
  my $expectedText = "John Doe";      #
                                      #
  AssertThat->document($pdfUnderTest) # the document under test
            ->and($reference)         # the reference document for the comparison
            ->restrictedTo($page2)    #
            ->hasText()               #
            ->...                     #
  ;                                   #
} "comparing a PDF with a reference"; #
```

❶ Das PDF-Dokument kann als Datentyp `String`, `File`, `InputStream`, `URL` oder `byte-Array` an die Funktion übergeben werden.

Wenn **mehrere PDF-Dokumente** in einen Test einfließen, beginnt er mit der Methode `AssertThat->eachDocument(..)`:

```
# Instantiation of PDFUnit for multiple documents:
...
my $pdfArray = [$pdfUnderTest1, $pdfUnderTest2];
AssertThat->eachDocument($pdfArray)    ❷ 5: „Mehrere Dokumente und Verzeichnisse“ (S. 13)
          ->hasFormat(A4_PORTRAIT)
;
...
```

❷ Die PDF-Dokumente können als `String[]`, `File[]`, `InputStream[]`, oder `URL[]` an die Funktion übergeben werden.

Tests können sich auch auf **alle PDF-Dokumente in einem Verzeichnis** beziehen. Solche Tests beginnen mit der Methode `AssertThat->eachDocument()->inFolder(..)`:

```
# Instantiation of PDFUnit for a folder:
...
AssertThat->eachDocument()
          ->inFolder($folder)              ❸ 5.3: „Verzeichnis testen“ (S. 14)
          ->passedFilter($allPdfunitFiles)
          -> ...
;
...
```

❸ Alle PDF-Dokumente in diesem Verzeichnis werden validiert. PDF-Dokumente in Unterverzeichnisse werden nicht geprüft.

Testbereiche

Die folgende Liste gibt einen vollständigen Überblick über die Testgebiete von PDFUnit. Die Kapitel sind aber nicht in diesem Handbuch beschreiben, sondern im Handbuch von PDFUnit-Java, letztendlich um Redundanzen zu vermeiden, die über kurz oder lang zu einer fehlerhaften Dokumentation führen würden. Hier der Link zum PDFUnit-Java Handbuch: http://www.pdfunit.com/de/documentation/java/.

```
# Every one of the following methods opens a new test scope:
#
# The detailed descriptions can be found in the manual of PDFUnit-Java (to avoid redundancy).
# The names of the methods are exactly the same.
#
# see http://www.pdfunit.com/de/documentation/java/
#
->asRenderedPage()

->containsImage(..)
->containsOneImageOf(..)

->hasAuthor()
->hasBookmark()
->hasBookmarks()
->hasCreationDate()
->hasCreator()
->hasEmbeddedFile(..)
->hasEncryptionLength(..)
->hasField(..)
->hasFields(..)
->hasFont()
->hasFonts()
->hasFormat(..)
->hasImage(..)
->hasJavaScript()
->hasJavaScriptAction()
->hasKeywords()
->hasLanguageInfo(..)
->hasLayer()
->hasLayers()
->hasLessPagesThan()
->hasLocalGotoAction()
->hasModificationDate()
->hasMorePagesThan()
->hasNamedDestination()

->hasNoAuthor()
->hasNoCreationDate()
->hasNoCreator()
->hasNoImage()
->hasNoKeywords()
->hasNoLanguageInfo()
->hasNoModificationDate()
->hasNoProducer()
->hasNoProperty(..)
->hasNoSubject()
->hasNoText()
->hasNoTitle()
->hasNoXFAData()
->hasNoXMPData()

->..  continued
```

```
... continuation:

->hasNumberOf...()

->hasOCG()
->hasOCGs()
->hasOwnerPassword(..)
->hasPermission()
->hasProducer()
->hasProperty(..)
->hasSignatureField(..)
->hasSignatureFields()
->hasSubject()
->hasText()
->hasTitle()
->hasUserPassword(..)
->hasVersion()
->hasXFAData()
->hasXMPData()
->hasZugferdData()

->haveSame...()

->isCertified()
->isCertifiedFor(..)
->isLinearizedForFastWebView()
->isSigned()
->isSignedBy(..)
->isTagged()

->restrictedTo(..)
```

Manche Testbereiche erreicht man erst nach einem weiteren Methodenaufruf:

```
# Validation of bar code and QR code:
->hasImage()->withBarcode()
->hasImage()->withQRcode()

# Validation based on Excel files:
->compliesWith()->constraints(excelRules)

# Validation of DIN5008 constraints:
->compliesWith()->din5008FormA()
->compliesWith()->din5008FormB()
->compliesWith()->pdfStandard()

# Validation around ZUGFeRD:
->compliesWith()->zugferdSpecification(..)
```

Hier noch einmal der Link zum PDFUnit-Java Handbuch: http://www.pdfunit.com/de/documentation/java/.

PDFUnit wird ständig weiterentwickelt und die Dokumentation aktuell gehalten. Sollten Sie Tests vermissen, schicken Sie Ihre Wünsche und Vorschläge an info[at]pdfunit.com.

Kapitel 4. Vergleiche gegen ein Referenz-PDF

4.1. Überblick

Viele Tests folgen dem Prinzip, ein einmal getestetes PDF-Dokument als Referenz für neu erstellte Dokumente zu benutzen. Solche Tests sind sinnvoll, wenn Prozesse, die das PDF erstellen, geändert werden, das Ergebnis aber unverändert bleiben soll.

Initialisierung

Die Instantiierung eines Referenz-Dokumentes erfolgt über die Methode `->and(..)`:

```
#
# Test: compare text with reference document
#
lives_ok {
  my $filenameTest   = "$resources_dir/master/compareToMaster_encrypted.pdf";
  my $filenameMaster = "$resources_dir/master/compareToMaster.pdf";
  my $userPassword   = 'user-password';
  AssertThat->document($filenameTest, $userPassword)   ❶
            ->and($filenameMaster)                     ❷
            ->restrictedTo(FIRST_PAGE)
            ->haveSameText()
  ;
} "compare text with reference document";
```

❶ Das Test-Dokument ist verschlüsselt und wird mit dem Passwort geöffnet.

❷ Das Referenz-Dokument ist hier nicht verschlüsselt. Falls es verschlüsselt wäre, muss das Passwort der Methode `->and()` als zweiter Parameter übergeben werden.

Passwörter dienen nur zum Öffnen der Dokumente, die Tests werden von dem Passwort nicht beeinflusst.

Überblick

Die folgende Liste gibt einen vollständigen Überblick über die vergleichenden Tests von PDFUnit. Links führen zu Kapiteln, die den jeweiligen Test ausführlich beschreiben.

Die folgende Liste gibt einen vollständigen Überblick über die vergleichenden Testgebiete von PDFUnit. Die Kapitel sind aber nicht in diesem Handbuch beschreiben, sondern im Handbuch von PDFUnit-Java, letztendlich um Redundanzen zu vermeiden, die über kurz oder lang zu einer fehlerhaften Dokumentation führen würden. Hier der Link zum PDFUnit-Java Handbuch: http://www.pdfunit.com/de/documentation/java/.

```
# Methods to compare two PDF documents:
#
# The detailed descriptions can be found in the manual of PDFUnit-Java (to avoid redundancy).
# The names of the methods are exactly the same.
#
# see http://www.pdfunit.com/de/documentation/java/
#

->haveSameAccessPermission()
->haveSameAccessPermission(..)
->haveSameAppearance()
->haveSameAuthor()
->haveSameBookmarks()
->haveSameCreationDate()
->haveSameCreator()
->haveSameEmbeddedFiles(..)
->haveSameFieldsByName()
->haveSameFieldsByValue()
->haveSameFormat()
->haveSameImages()
->haveSameJavaScript()
->haveSameKeywords()

...  continued
```

```
... continuation:

->haveSameLanguageInfo()
->haveSameLayerNames()
->haveSameModificationDate()
->haveSameNamedDestinations()
->haveSameNumberOfBookmarks()
->haveSameNumberOfEmbeddedFiles()
->haveSameNumberOfFields()
->haveSameNumberOfImages()
->haveSameNumberOfLayers()
->haveSameNumberOfNamedDestinations()
->haveSameNumberOfPages()
->haveSameProducer()
->haveSameProperties()
->haveSameProperty(..)
->haveSameSubject()
->haveSameTaggingInfo()
->haveSameText()
->haveSameTitle()
->haveSameXFAData()
->haveSameXMPData()
```

Hier noch einmal der Link zum PDFUnit-Java Handbuch: http://www.pdfunit.com/de/documentation/java/.

Kapitel 5. Mehrere Dokumente und Verzeichnisse

5.1. Überblick

Für die Mengen-Tests stehen fast alle Testmethoden zur Verfügung, die auch für Tests mit einzelnen PDF-Dokumenten existieren. Die folgende Liste zeigt die Methoden, die sowohl für ein ganzes Verzeichnis, als auch für ein angegebene Dokumentenmenge verwendet werden können. Eine Detailbeschreibung der jeweiligen Testmethoden finden Sie im Handbuch von PDFUnit-Java (um Redundanzen zu vermeiden, die über kurz oder lang zu einer fehlerhaften Dokumentation führen würden). Hier der Link zum PDFUnit-Java Handbuch: http://www.pdfunit.com/de/documentation/java/.

```
# Methods to validate a set of PDF documents:
#
# The detailed descriptions can be found in the manual of PDFUnit-Java (to avoid redundancy).
# The names of the methods are exactly the same.
#
# see http://www.pdfunit.com/de/documentation/java/
#

->compliesWith()
   ->constraints(..)
   ->din5008FormA()
   ->din5008FormB()
   ->pdfStandard()
   ->zugferdSpecification(..)

->containsOneImageOf(..)
->hasAuthor()
->hasBookmark()
->hasBookmarks()
->hasEncryptionLength(..)
->hasField(..)
->hasFields()
->hasFont()
->hasFonts()
->hasFormat(..)
->hasImage()
   ->withBarcode()
   ->withQRcode()
->hasJavaScript()
->hasKeywords()
->hasLanguageInfo(..)
->hasNoAuthor()
->hasNoImage()
->hasNoKeywords()
->hasNoLanguageInfo()
->hasNoProperty()
->hasNoSubject()
->hasNoText()
->hasNoTitle()
->hasNoXFAData()
->hasNoXMPData()

->hasNumberOf...()

->hasOwnerPassword(..)
->hasPermission()
->hasProperty(..)
->hasSignatureField(..)
->hasSignatureFields()
->hasSubject()
->hasText(..)
->hasTitle()
->hasUserPassword(..)

...  continued
```

```
... continuation:

->hasVersion()
->hasXFAData()
->hasXMPData()
->hasZugferdData()
->isCertified()
->isCertifiedFor(..)
->isLinearizedForFastWebView()
->isSigned()
->isSignedBy(..)
->isTagged()

->passedFilter(..)
```

Ein Test auf mehrere Dokumente oder Verzeichnisse bricht mit dem ersten fehlerhaften Dokument ab.

Die nächsten beiden Kapitel zeigen Beispiele für Tests mit einer Dokumentenmenge und mit einem Verzeichnis.

5.2. Mehrere Dokumente testen

Dieses Beispiel prüft, ob alle drei übergebenen Dokumente zwei bestimmte Textstücke enthalten:

```
lives_ok {
    my $pdf1 = "$resources_dir/document_en.pdf";
    my $pdf2 = "$resources_dir/document_es.pdf";
    my $pdf3 = "$resources_dir/document_de.pdf";
    my $pdfArray = [$pdf1, $pdf2, $pdf3];

    my $expectedDate = "28.09.2014";
    my $expectedDocumentID = "XX-123";

    AssertThat->eachDocument($pdfArray)
              ->restrictedTo(FIRST_PAGE)
              ->hasText()
              ->containing($expectedDate)
              ->containing($expectedDocumentID)
    ;
} "same text in multiple document";
```

Die PDF-Dokumente werden hier als `String[]` übergeben. Weiterhin werden die Typen `File[]`, `InputStream[]` und `URL[]` unterstützt.

5.3. Verzeichnis testen

Tests mit Verzeichnissen beginnen alle mit folgenden Methoden:

```
# Instantiation of PDFUnit for a folder:

AssertThat->eachDocument()                  ❶
          ->inFolder($folder)               ❷
          ->passedFilter($allPdfunitFiles)  ❸
          ->...
;
```

❶❷ Diese beiden Methoden sind der syntaktische Einstieg für Tests, die sich auf alle Dateien im angegebenen Verzeichnis beziehen, die die Dateiendung `PDF` haben. Der Vergleich auf die Dateiendung erfolgt case-insensitiv.

❸ Mit einem (optionalen) Filter kann die Menge der zu testenden Dokumente eingeschränkt werden. Es können mehrere Filter verwendet werden. Ohne Filter werden alle PDF-Dokumente des Verzeichnisses getestet.

Ein Test bricht ab, sobald ein Dokument im Verzeichnis den Test nicht erfüllt.

Falls die Verwendung von Filtern dazu führt, dass kein Dokument mehr übrig bleibt, wird eine Fehlermeldung erzeugt. Ebenso gibt es einen Fehler, wenn das Verzeichnis keine Dokumente enthält.

Beispiel - Alle Dokumente im Verzeichnis validieren

Das folgende Beispiel testet, ob alle PDF-Dokumente im angegebenen Verzeichnis die ZUGFeRD-Spezifikation erfüllen:

```
lives_ok {
  my $folder = File->new("$resources_dir/zugferd10/");

  AssertThat->eachDocument()
            ->inFolder($folder)
            ->compliesWith()
            ->zugferdSpecification(ZugferdVersion::VERSION10)
  ;
} "validate compliance with ZUGFeRD specification of all PDF of a given folder";
```

Beispiel - Ausgewählte Dokumente im Verzeichnisse validieren

Das nächste Beispiel überprüft den Titel aller PDF-Dokumente eines Verzeichnisses, die 'doc-under-test' oder 'reference' im Namen enthalten:

```
lives_ok {
  my $folder = File->new($resources_dir);
  my $allPdfunitFiles =
      FilenameMatchingFilter->new('.*(doc-under-test|reference)\.pdf$');

  AssertThat
      ->eachDocument()
      ->inFolder($folder)
      ->passedFilter($allPdfunitFiles)
      ->hasProperty("Title")->equalsTo("PDFUnit - Automated PDF Tests")
      ;
} "demo using a folder with a file filter";
```

Es gibt zwei Arten von Filtern. Der Filter `FilenameContainingFilter` untersucht, ob ein Dateiname einen bestimmten String enthält, und der Filter `FilenameMatchingFilter` wendet einen Regulären Ausdruck auf den Dateinamen an. Wichtig: In beiden Fällen bezieht sich die Analyse des Dateinamens auf den vollständigen Namen, d.h. auch auf den Pfad. Deswegen sollte beispielsweise ein Regulärer Ausdruck immer mit '`.*`' beginnen.

Kapitel 6. Unicode

Unicode in PDF

Funktionieren die bisher beschriebenen Tests auch mit Inhalten, die nicht ISO-8859-1 sind, beispielsweise mit russischen, griechischen oder chinesischen Texten und Metadaten?

Eine schwierige Frage. Denn auch wenn bei der Entwicklung von PDFUnit viel Wert darauf gelegt wurde, generell mit Unicode zu funktionieren, kann eine pauschale Antwort nur gegeben werden, wenn die eigenen Tests für PDFUnit selber mit „allen“ Möglichkeiten durchgetestet wurde. PDFUnit hat zwar etliche Tests für griechische, russische und chinesische Dokumente, aber es fehlen noch Tests mit hebräischen und japanischen PDF-Dokumenten. Insofern kann die eingangs gestellte Frage nicht abschließend beantwortet werden.

Prinzipiell tun Sie gut daran, sämtliche Werkzeuge auf UTF-8 zu konfigurieren, wenn Sie Unicode-Daten verarbeiten müssen.

Die folgenden Tipps im Umgang mit UTF-8 Dateien lösen nicht nur Probleme im Zusammenhang mit PDFUnit. Sie sind sicher auch in anderen Situationen hilfreich.

Einzelne Unicode-Zeichen

Metadaten und Schlüsselwörter können Unicode-Zeichen enthalten. Wenn Ihre Entwicklungsumgebung die fremden Fonts nicht unterstützt, können Sie ein Unicode-Zeichen in Perl mit `\x{nnnn}` innerhalb eines Double-Quoted-String schreiben, wie hier das Copyright-Zeichen „©“ als `\x{00A9}`:

```
#
# The document info 'producer' contains the copyright as a Unicode charactere.
#
lives_ok {
  my $pdfUnderTest = "$resources_dir/unicode/unicode_producer.pdf";
  AssertThat->document($pdfUnderTest)
            ->hasProducer()
            ->equalsTo("txt2pdf v7.3 \x{00A9} SANFACE Software 2004")  # 'copyright'
  ;
} "value of producer contains Unicode";
```

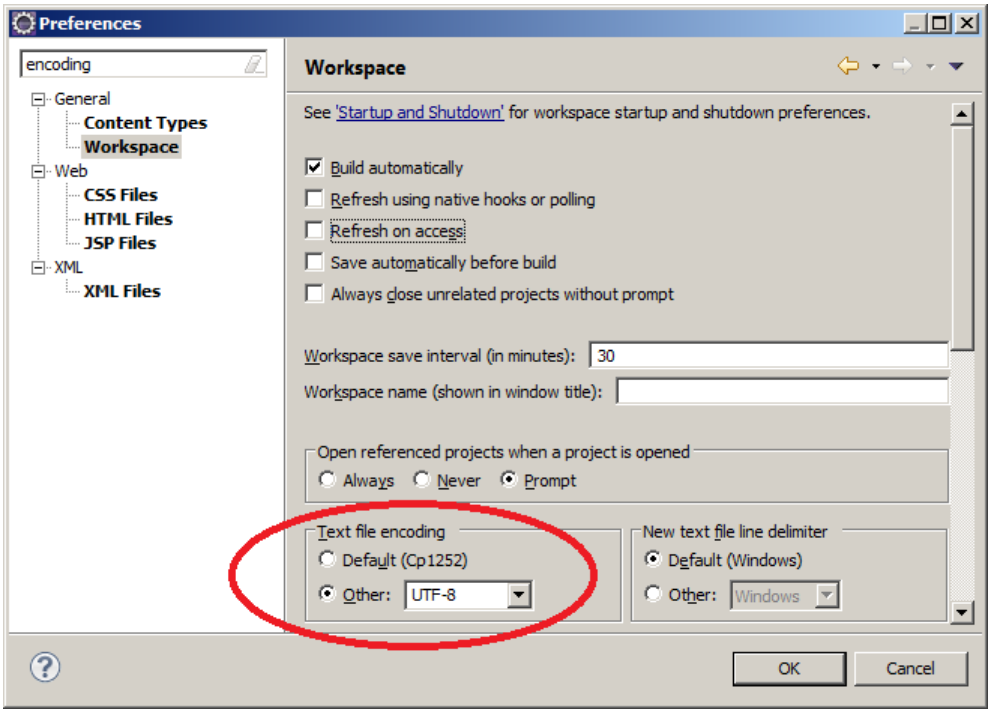
Längere Unicode-Texte

Selbstverständlich können Sie im Perl-Code direkt Unicode verwenden. Dann dürfen Sie aber nicht die `use utf8;` Anweisung vergessen. Das Test mit Unicode sieht dann so aus:

```
#
#  The document info 'subject' contains Unicode.
#
lives_ok {
  my $pdfUnderTest = "$resources_dir/unicode/unicode_subject.pdf";
  my $expectedSubject = "Εργαστήριο Μηχανικής ΙΙ ΤΕΙ ΠΕΙΡΑΙΑ / Μηχανολόγοι";
  AssertThat->document($pdfUnderTest)
            ->hasSubject()
            ->equalsTo($expectedSubject)
  ;
} "test subject with Greek characters";
```

Eclipse auf UTF-8 einstellen

Wenn Sie XML-Dateien in Eclipse erstellen, ist es nicht unbedingt nötig, Eclipse auf UTF-8 einzurichten, denn XML-Dateien sind auf UTF-8 voreingestellt. Für andere Dateitypen ist aber die Codepage des Betriebssystems voreingestellt. Sie sollten daher, wenn Sie mit Unicode-Daten arbeiten, das Default-Encoding für den gesamten Workspace auf UTF-8 einstellen:



Abweichend von dieser Standardeinstellung können einzelne Dateien in einem anderen Encoding gepeichert werden.

Unicode für unsichtbare Zeichen -

Im praktischen Betrieb trat einmal ein Problem auf, bei dem ein „non-breaking space“ in den Testdaten enthalten war, das zunächst als normales Leerzeichen wahrgenommen wurde. Der String-Vergleich lieferte aber einen Fehler, der erst durch die Verwendung von Unicode beseitigt werden konnte:

```
#
# String ends with NBSP.
#
lives_ok {
  my $pdfUnderTest = "$resources_dir/unicode/xfaBasicToggle.pdf";
  my $defaultNS = DefaultNamespace->new("http://www.w3.org/1999/xhtml");
  my $nodeValue = "The code for creating the toggle behavior involves switching "
                . "the border between raised and lowered, and maintaining the button's";
  my $nodeValueWithNBSP = $nodeValue . "\x{00A0}"; # The content terminates with a NBSP.
  my $nodeP7 = XMLNode->new("default:p[7]", $nodeValueWithNBSP, $defaultNS);
  AssertThat->document($pdfUnderTest)
            ->hasXFAData()
            ->withNode($nodeP7)
  ;
} "check for invisible blank (nbsp)";
```

Kapitel 7. Hilfsprogramme zur Testunterstützung

7.1. Allgemeine Hinweise für alle Hilfsprogramme

PDFUnit stellt Hilfsprogramme zur Verfügung, die Teilinformationen von PDF-Dokumenten in Dateien extrahieren, die anschließend in Tests genutzt werden können. Diese Programme sind aber nicht in diesem Handbuch beschreiben, sondern im Handbuch von PDFUnit-Java, letztendlich um Redundanzen zu vermeiden, die über kurz oder lang zu einer fehlerhaften Dokumentation führen würden. Hier der Link zum PDFUnit-Java Handbuch: http://www.pdfunit.com/de/documentation/java/. Folgende Hilfsprogramme stehen zur Verfügung:

```
# Utility programs belonging to PDFUnit:
#
# The detailed descriptions can be found in the manual of PDFUnit-Java (to avoid redundancy).
# The names of the methods are exactly the same.
#
# see http://www.pdfunit.com/de/documentation/java/
#

ConvertUnicodeToHex
ExtractBookmarks
ExtractEmbeddedFiles
ExtractFieldInfo
ExtractFontInfo
ExtractImages
ExtractJavaScript
ExtractNamedDestinations
ExtractSignatureInfo
ExtractXFAData
ExtractXMPData
ExtractZugferdData
RenderPdfPageRegionToImage
RenderPdfToImages
```

Die Hilfsprogramme erzeugen Dateien, deren Namen sich aus dem der jeweiligen Eingabedatei ableiten. Damit es keine Namenskonflikte mit eventuell bestehenden Dateien gibt, gelten diese Namenskonventionen:

- Die Namen beginnen mit einem Unterstrich.
- Die Namen besitzen zwei Suffixe. Das vorletzte lautet `.out`, das letzte ist der übliche Dateityp.

Beispielsweise wird aus der Datei `foo.pdf` die Ausgabe `_bookmarks_foo.out.xml` erzeugt. Benennen Sie sie um, wenn Sie diese Datei in Ihren Tests verwenden.

In den folgenden Kapiteln werden Batchdateien abgebildet, die zeigen, wie die Programme gestartet werden. Die Batchdateien sind Teil des Releases. Sie müssen aber einige der Inhalte, nämlich Classpath, Eingabedatei und Ausgabeverzeichnis an Ihre projektspezifischen Gegebenheiten anpassen.

Werden die Programme fehlerhaft gestartet, wird auf der Konsole ein Hilfetext mit der vollständigen Aufrufsyntax angezeigt.

Kapitel 8. Installation, Konfiguration, Update

8.1. Technische Voraussetzungen

Die Installation von PDF::PDFUnit folgt den Gepflogenheiten von CPAN-Moduln. Die Dokumentation ist auf http://search.cpan.org/dist/PDF-PDFUnit/ ausführlich beschrieben. Zusätzlich existiert eine ebenso ausführliche Beschreibung als README-Datei im Release.

PDF::PDFUnit verwendet seinerseits PDFUnit-Java. Um Redundanzen auf Dokumentationsebene zu vermeiden, wird hier auf das Kapitel 'Installation, Konfiguration, Update' im Handbuch von PDFUnit-Java (http://www.pdfunit.com/de/documentation/java/) verwiesen.

Kapitel 9. PDFUnit für Nicht-Perl Systeme

9.1. Kurzer Blick auf PDFUnit-Java

„PDFUnit-Java“ ist die erste Implementierung von PDFUnit und auch die Basis für die Implementierung in anderen Programmiersprachen. Für die XML-Implementierung ist PDFUnit-Java auch die technische Laufzeitumgebung. Sofern es möglich ist, werden die Schlüsselwörter aller Implementierungen von PDFUnit gleichlautend zu den Schlüsselwörtern in PDFUnit-Java gewählt.

Die API folgt dem „Fluent Interface“ (http://de.wikipedia.org/wiki/Fluent_Interface), wie die folgenden Beispiele zeigen:

```
@Test
public void hasTextOnFirstPageInPageRegion() throws Exception {
  String filename = "documentUnderTest.pdf";

  int leftX  =  50;
  int upperY = 130;
  int width  = 170;
  int height =  25;
  PageRegion pageRegion = new PageRegion(leftX, upperY, width, height);

  AssertThat.document(filename)
            .restrictedTo(FIRST_OAGE)
            .restrictedTo(pageRegion)
            .hasText()
            .containing("Content on first page")
  ;
}
```

```
@Test
public void compareFields() throws Exception {
  String filenameTest = "documentUnderTest.pdf";
  String filenameReference = "reference.pdf";

  AssertThat.document(filenameTest)
            .and(filenameReference)
            .haveSameFieldsByName()
            .haveSameFieldsByValue()
  ;
}
```

```
@Test
public void hasSignature() throws Exception {
  String filename = "documentUnderTest.pdf";
  Calendar signingDate = DateHelper.getCalendar("2007-10-14", "yyyy-MM-dd");

  AssertThat.document(filename)
            .hasSignatureField("sign_rbl")
            .signedBy("Raymond Berthou")
            .signedOn(signingDate)
  ;
}
```

PDFUnit-Java ist in einer eigenen Dokumentation ausführlich beschrieben. Siehe .

9.2. Kurzer Blick auf PDFUnit-XML

Tester müssen keine Java-Kenntnisse besitzen, um PDF-Dokumente automatisiert zu testen. Für eine auf XML basierende Systemlandschaft gibt es unter der Bezeichnung 'PDFUnit-XML' Laufzeitkomponenten, Skripte, XML Schema und Stylesheets zum Testen von PDF-Dokumenten. Die Funktionalität ist voll kompatibel zu 'PDFUnit-Java'.

Die folgenden Beispiele geben einen Einblick in PDFUnit-XML:

```
<testcase name="hasTextOnSpecifiedPages_Containing">
  <assertThat testDocument="content/diverseContentOnMultiplePages.pdf">
    <hasText onPage="1, 2, 3" >
      <containing>Content on</containing>
    </hasText>
  </assertThat>
</testcase>
```

```
<testcase name="hasTitle_MatchingRegex">
  <assertThat testDocument="documentInfo/documentInfo_allInfo.pdf">
    <hasTitle>
      <startingWith>PDFUnit sample</startingWith>
      <matchingRegex>.*Unit.*</matchingRegex>
    </hasTitle>
  </assertThat>
</testcase>
```

```
<testcase name="compareText_InPageRegion">
  <assertThat testDocument="test/test.pdf"
              referenceDocument="reference/reference.pdf"
  >
    <haveSameText on="EVERY_PAGE" >
      <inRegion upperLeftX="50" upperLeftY="720" width="150" height="30" />
    </haveSameText>
  </assertThat>
</testcase>
```

```
<testcase name="hasField_MultipleFields">
  <assertThat testDocument="acrofields/simpleRegistrationForm.pdf">
    <hasField withName="name" />
    <hasField withName="address" />
    <hasField withName="postal_code" />
    <hasField withName="email" />
  </assertThat>
</testcase>
```

Die Namen der Tags und Attribute stimmen überwiegend mit der Java-API überein und folgen ebenfalls der Idee des 'Fluent Interfaces' (http://de.wikipedia.org/wiki/Fluent_Interface).

Die XML-Syntax ist mit passenden XML Schema Dateien abgesichert.

Eine genaue Beschreibung steht als eigenständige Dokumentation zur Verfügung.

9.3. Kurzer Blick auf PDFUnit-NET

Als 'PDFUnit-NET' steht PDFUnit auch für .NET-Anwendungen seit Dezember 2015 zur Verfügung.

```
[TestMethod]
public void HasAuthor()
{
  String filename = "documentUnderTest.pdf";
  AssertThat.document(filename)
            .hasAuthor()
            .matchingExact("PDFUnit.com")
  ;
}
```

```
[TestMethod]
[ExpectedException(typeof(PDFUnitValidationException))]
public void HasAuthor_StartingWith_WrongString()
{
  String filename = "documentUnderTest.pdf";
  AssertThat.document(filename)
            .hasAuthor()
            .startingWith("wrong_sequence_intended")
  ;
}
```

Die Kompatibilität zu PDFUnit-Java wird dadurch erreicht, dass aus der Java-Version eine DLL generiert wird. Das hat allerdings zur Folge, dass die Methodennamen in C# mit Kleinbuchstaben beginnen.

Für PDFUnit-NET existiert eine eigene Dokumentation.

Kapitel 10. Typische Fehler

10.1. Übersicht über typische Fehler

Jeder Entwickler macht Fehler. Viele Entwickler machen die gleichen Fehler. Wer Fehler kennt, findet die Lösung schneller. Deshalb soll dieses Kapitel helfen, typische Fehler mit ihren Lösungen bekannt zu machen.

Liste der beschriebenen Fehler:

```
# Overview of typical errors:

10.2: „Datei nicht gefunden“ (S. 22)
10.3: „Java Syntaxfehler wegen Tippfehler“ (S. 22)
10.4: „Fehlerhafter Konstruktoraufruf“ (S. 23)
10.5: „Arrays für Java-varargs“ (S. 23)
10.6: „Ein Parameter ist 'null'“ (S. 24)
10.7: „Java-'Punkt' statt Perl-'Pfeil'“ (S. 25)
```

10.2. Datei nicht gefunden

Fehlermeldung

```
# PDF document 'C:\daten\...\resources\XXX.pdf' could not be loaded.
```

Erklärung

PDF Dokumente und andere Dateien eines Tests müssen in einem Pfad liegen, den der Java-Prozess findet.

Beispiel mit Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/XXX.pdf";             # error, file does not exist
  AssertThat->document($pdfUnderTest)
            ->hasText()
  ;
} "typical error, file not found";;
```

Beispiel ohne Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/doc-under-test.pdf";  #ok
  AssertThat->document($pdfUnderTest)
            ->hasText()
  ;
} "no error, file exists";
```

10.3. Java Syntaxfehler wegen Tippfehler

Fehlermeldung

```
# Failed test 'error intended, wrong method name'
# died: No public method 'hasLanguageXXX' defined for
# class 'main::com::pdfunit::validators::DocumentValidator'
# at C:/.../pdfunit-typical-error_java-syntax-error.t line 36.
Can't locate object method "getMessage" via package.
"No public method 'hasLanguageXXX' defined for
class 'main::com::pdfunit::validators::DocumentValidator'
at C:/.../pdfunit-typical-error_java-syntax-error.t line 36.
```

Erklärung

Das Modul Inline::Java sucht in der entsprechenden Java-Klasse nach der Methode mit dem falsch geschriebenen Namen, findet sie nicht und gibt dann eine Fehlermeldung aus. Diese Fehlermeldung enthält den Namen der Klasse, in der gesucht wurde. Und für diese Klasse sollten Sie die Java-doc-Dokumentation heranziehen, um dem Tippfehler auf die Spur zu kommen.

Beispiel mit Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/language/localeDemo_de.pdf";
  AssertThat->document($pdfUnderTest)
            ->hasLanguageXXX('de')     # error, not existing method
  ;
} "typical error, wrong method name";
```

Beispiel ohne Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/language/localeDemo_de.pdf";
  AssertThat->document($pdfUnderTest)
            ->hasLanguageInfo('de')    # syntax OK
  ;
} "no error, correct method name";
```

10.4. Fehlerhafter Konstruktoraufruf

Fehlermeldung

```
# died: Undefined subroutine &main::com::pdfunit::filter::region::PageRegion
# called at C:/.../pdfunit-typical-error_incorrect-use-of-constructor.t line 36.
```

Erklärung

Der Konstruktor einer Java-Klasse muss mit dem Methodennamen „new“ aufgerufen werden. Falls das vergessen wird, erscheint diese Fehlermeldung.

Beispiel mit Fehler

```
lives_ok {
  my $ulX    =   0;
  my $ulY    =   0;
  my $width  = 210;
  my $height =  50;
  my $headerRegion =PageRegion->($ulX, $ulY, $width, $height);  # syntax error, missing 'new'

} "typical error, incorrect constructor syntax";
```

Beispiel ohne Fehler

```
lives_ok {
  my $ulX    =   0;
  my $ulY    =   0;
  my $width  = 210;
  my $height =  50;
  my $headerRegion =PageRegion->new($ulX, $ulY, $width, $height); # ok

} "no error, correct constructor syntax";
```

10.5. Arrays für Java-varargs

Fehlermeldung

```
# died: Wrong number of arguments at C:/Perl64/site/lib/Inline/Java/Object.pm line 107.
```

Erklärung

Ein Java-Methode kann eine variable Anzahl an Parametern haben. An solche Methoden muss Perl ein Array übergeben.

Beispiel mit Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/doc-under-test.pdf";
  my $pages134 = PagesToUse->getPages(1, 3, 4);      #  syntax error
  AssertThat->document($pdfUnderTest)
            ->restrictedTo($pages134)
            ->hasText()
  ;
} "typical error, no array for Java varargs";
```

Beispiel ohne Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/doc-under-test.pdf";
  my $pages134 = PagesToUse->getPages([1, 3, 4]);    # ok
  AssertThat->document($pdfUnderTest)
            ->restrictedTo($pages134)
            ->hasText()
  ;
} "no error, array used for Java varargs";
```

10.6. Ein Parameter ist 'null'

Fehlermeldung

```
# Failed test 'typical error, parameter is null'
# died: main::com::pdfunit::errors::PDFUnitValidationException=HASH(0x3b6fc30)
# Invalid argument. One parameter is null.
```

Erklärung

Vielleicht wurde die Wertzuweisung nur vergessen oder ist während eines Refactorings nachträglich verloren gegangen. Auf der Java-Seite werden Null-Parameter erkannt und mit der gezeigten Fehlermeldung abgewiesen.

Beispiel mit Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/language/localeDemo_de.pdf";
  my $language;                              # null value intended
  AssertThat->document($pdfUnderTest)
            ->hasLanguageInfo($language)    # error, parameter is null
  ;
} "typical error, parameter is null";
```

Beispiel ohne Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/language/localeDemo_de.pdf";
  my $languageDE = 'de';                       # OK
  AssertThat->document($pdfUnderTest)
            ->hasLanguageInfo($languageDE)
  ;
} "no error, parameter is not null";
```

10.7. Java-'Punkt' statt Perl-'Pfeil'

Fehlermeldung

```
# Failed test 'error intended, 'Java point' used'
# died: Undefined subroutine &main::hasText called
```

Erklärung

Wenn Sie ein Code-Beispiel aus der Java-Dokumentation kopieren und nicht vollständig in die Perl-Syntax übertragen, kommt es zu diesem Fehler. Ersetzen Sie den 'Punkt' von Java, der vor einem Methodennamen steht, durch einen 'Pfeil' in Perl.

Beispiel mit Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/helloworld.pdf";
  AssertThat->document($pdfUnderTest)
            .hasText()                # syntax error, don't use a 'point' here
  ;
} "typical error, 'Java point' used";
```

Beispiel ohne Fehler

```
lives_ok {
  my $pdfUnderTest = "$resources_dir/helloworld.pdf";
  AssertThat->document($pdfUnderTest)
            ->hasText()               # ok
  ;
} "no error, 'Perl arrow' used";
```

Kapitel 11. Anhang

11.1. Instantiierung der PDF-Dokumente

An dieser Stelle sei wiederum auf das Handbuch von PDFUnit-Java (http://www.pdfunit.com/de/documentation/java/) verwiesen. In dessen Anhang werden verschiedene Aspekte rund um das Testen von PDF-Dokumenten erklärt. Eine Wiederholung wäre eine unerwünschte Redundanz auf Dokumentationsebene. Die Namen von Methoden und Konstanten lauten gleich, insofern werden Perl-Benutzer keine Probleme haben, die Syntax von Java auf die Syntax von Perl zu übertragen.

Hier die Themen, die im Anhang von PDFUnit-Java behandelt werden:

```
# Inhalt des Anhangs der Dokumentation von PDFUnit-Java
#
# siehe http://www.pdfunit.com/de/documentation/java/
#

- Instantiierung der PDF-Dokumente
- Seitenauswahl
- Seitenausschnitt definieren
- Textvergleich
- Behandlung von Whitespaces
- Anführungszeichen in Suchbegriffen
- Datumsauflösung
- Maßeinheiten - Points und Millimeter
- Fehlermeldungen
- Sprache für Fehlermeldungen einstellen
- XPath-Einsatz
- JAXP-Konfiguration
- Versionshistorie
- Nicht Implementiertes, Bekannte Fehler
```

Stichwortverzeichnis