02-2015 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

JAVA IST SUPER STARK

Programmierung
JavaScript für Java-Entwickler

Cloud Computing
Software-Architekturen in wolkigen Zeiten

Applikationsserver
JBoss vs. WebLogic Server

JavaServer Faces
Interview mit Spec Lead Ed Burns

4 191978 304903 02

iJUG Verbund

In Xtend geschriebener Code ist meist kompakter und eleganter als sein Java-Äquivalent

Der Autor ist ein sehr großer Fan von Java, aber auch von C#. Er stellt einige interessante Seiten von C# vor

Die Korrektheit erzeugter PDF-Dokumente überprüfen – nicht manuell, sondern automatisiert

PDF-Dokumente automatisiert testen

Carsten Siedentop, pdfunit.com

Viele PDF-Dokumente sind heutzutage das Ergebnis fachlicher Abläufe. Verträge, monatliche Rechnungen aber auch Kataloge und andere Druckstücke entstehen durch individuelle Programme, die nur selten fehlerfrei sind. Da es manchmal schwierig ist, diese Programme zu testen, wäre es hilfreich, wenigstens die Korrektheit der erzeugten PDF-Dokumente zu überprüfen – nicht manuell, sondern automatisiert.

Die Anzahl der verfügbaren APIs für automatische Tests ist überschaubar. Als erstes sei PDFBox *(siehe „https://pdfbox.apache.org")* genannt, ein API für den Zugriff auf verschiedene Inhalte von PDF-Dokumenten. Es ist kein Test-Framework, bietet aber die Möglichkeit, eigene „assert()"-Methoden zu schreiben. Die aktuelle Version ist PDFBox 1.8.8 von Dezember 2014.

Zweitens gibt es das Projekt „jPdfUnit" *(siehe „http://jpdfunit.sourceforge.net")*. Es ist ein Aufsatz auf PDFBox und bietet „assert()"-Methoden für den Vergleich von Texten. Das Projekt ist inaktiv, momentan ist die Version jPdfUnit 1.2 von Dezember 2011 aktuell.

Als Drittes gibt es PDFUnit, ein Testwerkzeug, das entstand, weil die zuvor genannten APIs die benötigte Funktionalität nicht abdecken konnten. Die aktuelle Version ist 2014.06, die nächste ist für April 2015 angekündigt. Der Autor ist an der Entwicklung von PDFUnit beteiligt. Die nachfolgenden Abschnitte beschreiben die verfügbaren Funktionen und deren Verwendung.

Texte überprüfen

Das erste Code-Beispiel *(siehe Listing 1)* zeigt Validierungsmethoden auf erwartete Texte auf der ersten Seite des zu testenden PDF-Dokuments. Es wird der normale JUnit-Report erzeugt.

Die komplette API ist flüssig gebaut. Der Einstieg für alle Tests ist die statische Methode „AssertThat.document(pdfUnderTest)...". Alle umgangssprachlich benannten Methoden sind typisiert, sodass Eclipse nur die erlaubten Methoden als Line-Completion anzeigt *(siehe Abbildung 1)*. Insgesamt stehen zwölf Methoden zur Verfügung, um die Anwesenheit oder Abwesenheit von Text zu überprüfen. *Listing 2* zeigt einige davon.

```
@Test
public void hasContentOnFirstPage() {
  String filename = "documentUnderTest.pdf";
  AssertThat.document(filename)
     .hasContent(ON_FIRST_PAGE)
     .startingWith("Lorem")
     .containing("ipsum")
     .endingWith("est laborum.");
}
```

Listing 1: Text auf der ersten Seite

```
// Überprüfung der An- und Abwesenheit von Text, :
.hasContent(..).containing(.., WhitespaceProcessing)
.hasContent(..).endingWith(..)
.hasContent(..).matchingComplete(.., WhitespaceProcessing)
.hasContent(..).matchingRegex(..)
.hasContent(..).notContaining(.., WhitespaceProcessing)
.hasContent(..).notMatchingRegex(..)
.hasContent(..).startingWith(..)
// und weitere Methoden
```

Listing 2: Vergleichsmethoden für Text

Bei einer automatischen Dokumenten-Erstellung können Zeilenumbrüche nicht vorhergesehen werden. Deshalb bieten manche „assert"-Methoden die Möglichkeit, die Art der Whitespace-Behandlung von außen mitzugeben („ignore", „normalize", „keep"). Bei Tests auf mehrseitigen Dokumenten muss es eine Möglichkeit geben, Seiten zu benennen. In *Listing 1* ist der Test auf die erste Seite beschränkt. *Listing 3* zeigt weitere nützliche Konstanten für die Seitenauswahl. Darüber hinaus können mit „PagesToUse.getPages(1, 2, 3)" individuelle Seiten definiert werden *(siehe Listing 5)*.

Tests auf Seitenausschnitte begrenzen

Es kann notwendig sein, Tests auf Ausschnitte einer Seite zu beschränken. Soll beispielsweise Text nur in der Fußzeile überprüft werden, wird ein Rechteck mit der passenden Größe für diesen Seitenausschnitt definiert. Dieser Ausschnitt ist durch vier Eigenschaften definiert. Es gibt die x/y-Position für die linke obere Ecke des Ausschnitts auf der PDF-Seite sowie die Breite und Höhe des Ausschnitts. Die Standard-Einheit sind „Millimeter", es können aber auch „Zentimeter", „Inches" und „DPI72" angegeben werden *(siehe Listing 4)*.

Zwei PDF-Dokumente vergleichen

Textteile, Seitenzahlen, Bilder, Schriften oder auch das Layout ganzer Seiten oder von Sei-

```
ON_ANY_PAGE,   ON_EACH_PAGE
ON_EVEN_PAGES, ON_ODD_PAGES
ON_FIRST_PAGE, ON_LAST_PAGE
```

Listing 3: Einschränkungen von Tests auf definierten Seiten

tenausschnitten eines Testdokuments können mit entsprechenden Teilen eines Master-Dokuments verglichen werden. *Listing 5* zeigt den Vergleich der jeweils zweiten Seite zweier PDF-Dokumente als gerenderte Images. Sind die beiden Images nicht gleich, wird zusätzlich zur Fehlermeldung noch ein Differenzbild erstellt *(siehe Abbildung 2)*.

```
@Test
public void hasContentOnFirstPage_InClippingArea() {
  String filename = "documentUnderTest.pdf";
  double upperLeftX  = 17.6; // in millimeter
  double upperLeftY  = 45.8;
  double w        = 60.0; // width
  double h        = 8.8; // height
  ClippingArea inClippingArea = new
        ClippingArea(upperLeftX, upperLeftY, w, h);
  AssertThat.document(filename)
        .hasContent(ON_FIRST_PAGE, inClippingArea)
        .startingWith("Lorem")
        .containing("ipsum")
        .endingWith("est laborum.");
}
```

Listing 4: Text innerhalb eines Seitenausschnitts

```
@Test
public void comparePDFWithMasterAsRenderedImages_Page2() {
  String filenameTest = "documentUnderTest.pdf";
  String filenameMaster = "master.pdf";
  PagesToUse ON_PAGE_2 = PagesToUse.getPage(2);
  AssertThat.document(filenameTest)
     .and(filenameMaster)
     .haveSameAppearance(ON_PAGE_2);
}
```

Listing 5: Vergleich von Test-PDF und Master als Images

```
@Test
public void containsLogo_OnAllPagesAfter1() {
  String filename = „documentUnderTest.pdf";
  String imageFileName = "images/myCompanyLogo.png";
  File imageFile = new File(imageFileName);
  AssertThat.document(filename)
     .containsImage(imageFile, OnEveryPage.after(1));
}
```

Listing 6: Logo auf jeder Seite ab Seite 2

Wenn zwei Unternehmen fusionieren, ändert sich häufig das Logo auf den Rechnungen. Das ist kein Programmier-Problem. Dennoch geht manchmal etwas schief, und so sollte einfach getestet werden, ob es wirklich geklappt hat *(siehe Listing 6)*.

Unsichtbare Teile im PDF

Die bisherigen Tests könnten alle manuell ausgeführt werden, auch wenn das zeitaufwändig und fehleranfällig wäre. Unsichtbare Inhalte hingegen, wie Schriften, Lesezeichen, JavaScript, Formulare und XMP-Daten eines PDF-Dokuments lassen sich ausschließlich auf elektronischem Weg testen. So überprüft der Test in *Listing 7*, ob Feldnamen in Formularen eindeutig sind. *Listing 8* zeigt, wie man die Existenz von JavaScript innerhalb eines PDF-Dokuments überprüft.

XMP-Daten validieren

Ein sensibler Inhalt in PDF-Dokumenten sind XMP-Daten. „XMP" steht für „Extensible Metadata Platform" und ist ein Standard, um Metadaten in digitale Medien einzubetten. Häufig wird ein PDF-Dokument mit XMP-Daten angereichert, um diese zu einem späteren Zeitpunkt wieder auszuwerten. Insofern macht es Sinn, wenn die XMP-Daten auch richtig sind. Für die Validierung bietet PDFUnit die Funktion „hasXMPDate()" mit sehr flexiblen Nachfolgefunktionen, in denen XPath mit all seiner Mächtigkeit ausgereizt werden kann. Namespaces werden selbst-

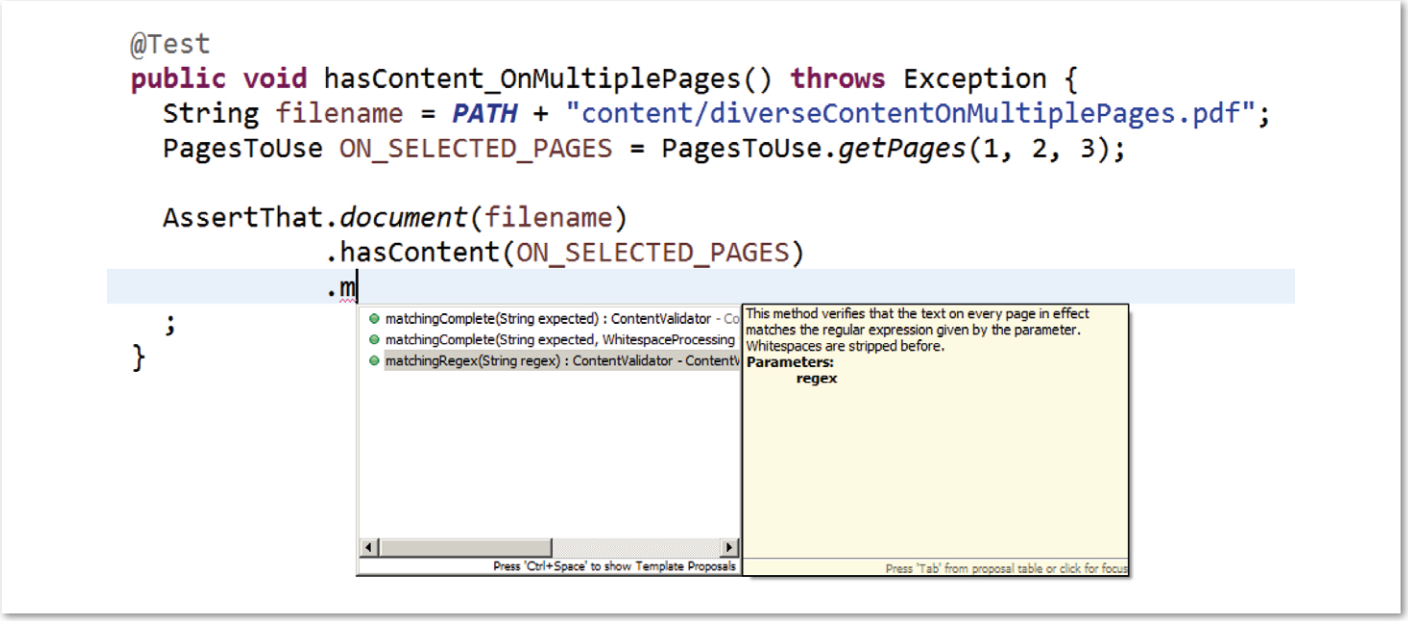

Abbildung 1: Unterstützung im Eclipse

```
@Test
public void hasFields_WithoutDuplicateNames() {
  String filename = "documentUnderTest.pdf";
  AssertThat.document(filename)
      .hasFields()
      .allWithoutDuplicateNames();
}
```

Listing 7: Formular-Felder ohne doppelte Namen

```
@Test
public void hasJavaScript() {
  String filename = "documentUnderTest.pdf";
  String scriptFile = "javascript/fieldValidations.js";
  InputStream scriptFile = new FileInputStream(scriptFile);
  AssertThat.document(filename)
      .hasJavaScript()
      .containing(scriptFile);
}
```

Listing 8: JavaScript in PDF überprüfen

```
@Test
public void hasXMPData_WithNodeAndValue() {
  String filename = "documentUnderTest.pdf";
  XMLNode nodeCreateDate =
  new XMLNode("xmp:CreateDate", "2011-02-08T15:04:19+01:00");
  XMLNode nodeModifyDate =
  new XMLNode("xmp:ModifyDate", "2011-02-08T15:04:19+01:00");

  AssertThat.document(filename)
      .hasXMPData()
      .withNode(nodeCreateDate)
      .withNode(nodeModifyDate);
}

@Test
public void hasXMPData_XPathExpression() {
  String filename = "ocumentUnderTest.pdf";
  DefaultNamespace ns = new
  DefaultNamespace("http://purl.org/dc/elements/1.1/");
  XPathExpression expr =
  new XPathExpression("//foo:format = 'application/pdf'", ns);

  AssertThat.document(filename)
      .hasXMPData()
      .matchingXPath(expr);
}
```

Listing 9: Zwei Tests auf XMP-Daten

verständlich unterstützt. *Listing 9* zeigt zwei Beispiele.

Nützliche kleine Tools

Es ist zwar schön, dass es Tests auf unsichtbare Inhalte gibt, es gibt aber noch ein prinzipielles Problem mit den Inhalten: Wenn Fehler auftauchen, will man erkennen, welche Inhalte ein PDF-Dokument tatsächlich hat. Zur Beantwortung dieser Frage bietet PDFUnit viele kleine Tools, *Listing 10* zeigt eine kleine Auswahl. Die Namen der Klassen sprechen für sich. Es sind Konsolenprogramme, die die notwendigen Daten als Parameter erwarten *(siehe Listing 11)*.

PDFUnit auch als XML-API

Damit auch Nicht-Java-Entwickler ihre PDF-Dokumente testen können, gibt es PDFUnit auch als XML-Schnittstelle. Sämtliche Funktionen der Java-Version stehen dabei über XML zur Verfügung. Die XML-Schnittstelle kommt mit einer XML-Schema-Definition, sodass ein XML-Editor die Erstellung der Tests gut unterstützen kann *(siehe Abbildung 3)*. Die Ausführung der Tests erzeugt denselben JUnit-Report wie normale Unit-Tests. *Listing 12* zeigt das Java-Beispiel aus *Listing 4* in seiner XML-Form.

Die Nachteile

PDFUnit ist nicht kostenlos. Entwickler sind es gewohnt, mit kostenlosen Produkten umzugehen, andererseits haben sie alle für Betriebssystem, Datenbank und Internetzugang bezahlt. In die Entwicklung von PDF-

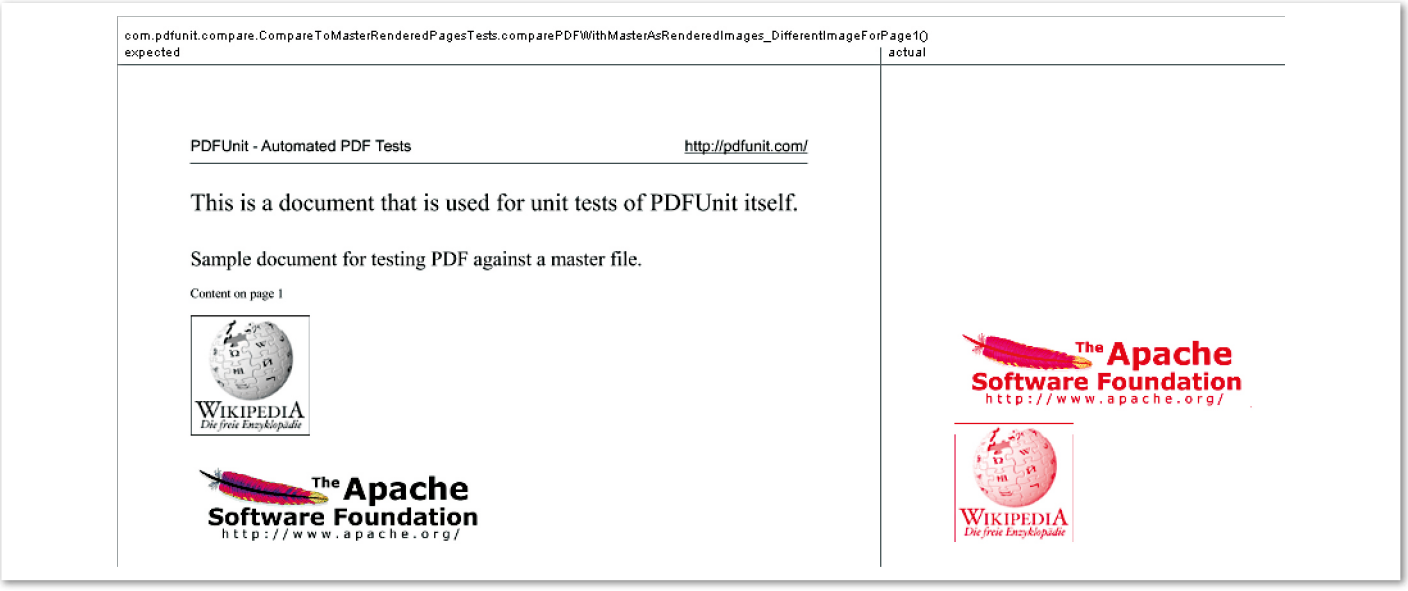

Abbildung 2: Diff-Image beim Seitenvergleich als gerenderte Seiten

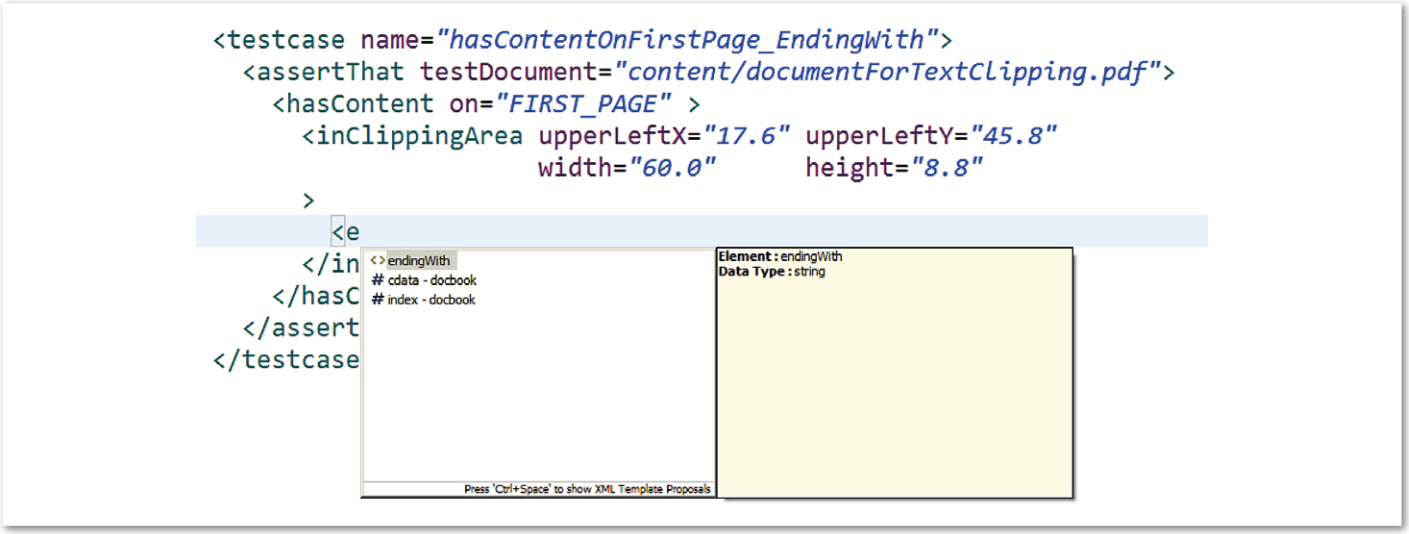

Abbildung 3: XML-Schnittstelle von PDFUnit im Eclipse-Editor

```
ExtractEmbeddedFiles
ExtractFieldsInfo
ExtractImages
ExtractJavaScript
ExtractSignaturesInfo
ExtractXMPData
RenderPdfClippingAreaToImage
```

Listing 10: Nützliche Tools rund um PDF (Auswahl)

```
set TOOL=com.pdfunit.tools.ExtractImages
set OUT_DIR=./tmp
set PASSWD=
set IN_FILE=documentUnderTest.pdf
java %TOOL%  %IN_FILE%  %OUT_DIR%  %PASSWD%
```

Listing 11: Extraktion von Bildern aus einem PDF-Dokument

```
<testcase name="hasContent_OnFirstPage_InClippingArea">
  <assertThat testDocument="documentUnderTest.pdf">
     <hasContent on="FIRST_PAGE" >
        <inClippingArea upperLeftX="17.6"
              upperLeftY="45.8"
              width="60.0"
              height="8.8"
        >
           <startingWith>Lorem</startingWith>
           <containing>ipsum</containing>
           <endingWith>est laborum.</endingWith>
        </inClippingArea>
     </hasContent>
  </assertThat>
</testcase>
```

Listing 12: Beispiel für PDFUnit-XML

Unit sind zwei Personenjahre an Entwicklungszeit geflossen. Solange kann niemand umsonst arbeiten. Immerhin sind während der Entwicklung nützliche Informationen an verschiedene Open-Source-Projekte zurückgeflossen.

Zukünftige Entwicklung

Neue Funktionen entstehen überwiegend durch Kundenanfragen. Parallel dazu wird das Projekt intern auf Java-8 (Streams) umgestellt, ohne dass dadurch die Schnittstelle verändert wird. Solche Änderungen können aufgrund der hohen Testabdeckung von PDFUnit problemlos durchgeführt werden. Das nächste Release erscheint im April 2015. Dann wird auch die Perl-API verfügbar sein, die ebenfalls zurzeit entwickelt wird. Ausführliche Dokumentationen und ein Beispielprojekt für Eclipse stehen unter „www.pdfunit.com“ zur Verfügung.

Fazit

Programme mögen kompliziert sein, aber die von Ihnen erzeugten PDF-Dokumente gehen an Kunden. Es ist einfach geworden, Dokumente zu testen – sie sollten fehlerfrei sein.

Carsten Siedentop
c.siedentop@pdfunit.com

Carsten Siedentop ist seit 23 Jahren freiberuflich als Anwendungsentwickler in Software-Projekten (Java, COBOL) tätig. Zusätzlich gibt er sein Wissen seit 18 Jahren als Trainer weiter. Im Leben jenseits des Computers nimmt die Musik einen großen Raum ein, unter anderem ist er Sänger an der Kölner Oper.

http://ja.ijug.eu/15/2/15