Automated PDF Tests with Perl

PDFUnit-Perl

Carsten Siedentop

Table of Contents

Preface

The Current Situation of Testing PDF in Projects

These days telephone bills, insurance policies, official notifications and many types of contract are delivered as PDF documents. They are the result of a process chain consisting of programs in various programming languages using numerous libraries. Depending on the complexity of the documents to be produced, such programming is not easy. The software may have errors and statistically will have errors. So it should be tested in some of the following ways:

- Is the expected text within the expected page region?
- Is the bar code's text the expected text?
- Does the layout fulfill the requirements?
- Do the embedded ZUGFeRD data have the expected values?
- Are the values of the embedded ZUGFeRD data correspond with the visible values?
- Does a PDF comply with DIN 5008?
- Is the PDF signed? When and by whom?

It should scare developers, project managers and CEO's that until now there is almost no way of repeatedly testing PDF documents. And even the options which are available are not used as frequently as they should be. Unfortunately, manual testing is widespread. It is expensive and prone to errors.

With PDFUnit, any document, whether created using a powerful design tool, exported from MS Word or LibreOffice, processed using an API, or dropped out of an XSL-FO workflow, can be tested.

Intuitive API

The interface of PDFUnit-Perl is a wrapper around PDFUnit-Java. The names of the methods try to follow the English language as closely as possible and thus support general thought patterns. The resulting code is easy to maintain.

The next examples shows how easy it is:

```
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"
);
```

It's not necessary for a test developer to have detailed knowledge about the structure of PDF or how it was created when he wants to write a test.

Time to Start

Don't gamble with the data and processes of your document processing system. Check the output of the PDF creating programs with automated tests.

Chapter 1. About this Documentation

Who Should Read it

This manual is for Perl programmers whose job it is to ensure that generated PDF documents are "correct".

It is assumed that you have a basic knowledge of Perl programming. Knowledge of test automation is not necessary, but helps to understand the text.

Code Examples

The code snippets in the following chapters use the module `Test::More`. But PDFUnit-Perl can also be used with other test modules. The distribution of `PDF::PDFUnit` (PDFUnit-Perl) contains some running examples.

References to the Java Documentation

This manual provides explanations and examples as much as possible for a good introduction into PDFUnit-Perl. But for a complete description you have to read the manual of PDFUnit-Java (http://www.pdfunit.com/en/documentation/java/index.html). Repeating all information here would be highly redundant. And because the method names are identical, Perl programmers can easily transfer the syntax of PDFUnit-Java to Perl.

API of PDFUnit-Java

The Javadoc documentation of the API is available online: http://www.pdfunit.com/api/javadoc/index.html.

Other Programming Languages

PDFUnit is available for Perl, Java, .NET and as an XML dialect. Separate documentation exists for each language.

If there are Problems

If you have problems to test a PDF, please search for a similar problem in the internet. Maybe, you find a solution. Finally, you are invited to write to info[at]pdfunit.com and describe the problem. We'll try to help you.

New Features Wanted?

Do you need other test functions? Please feel free to send your requirements to info@pdfunit.com. You are invited to influence the further development of PDFUnit.

Responsibility

Some examples in this book use PDF documents from the web. For legal reasons I make clear that I dissociate myself from their content, for instance I can not read Chinese. These documents support tests, for which I could not create my own test documents, e.g. the Chinese PDF documents.

Acknowledgement

Axel Miesen developed the Perl-API of PDFUnit and during that time he asked a lot of questions about the Java API. PDFUnit Java has profitted greatly from his input. Thank you, Axel.

Unfortunately, my English is not as good as I would like. But my colleague John Boyd-Rainey read the first version of this English documentation and corrected a huge number of misplaced commas and other typical errors. Thank you, John, for your perseverance and thoroughness. However, all remaining errors are my fault. He also asked critical questions which helped me to sharpen some descriptions.

Production of this Documentation

This documentation was created with DocBook-XML and both PDF and HTML are generated from one text source. It is well known that the layout can be improved in both formats, e.g. the pagebreaks in PDF format. And improving the layout is already on the to-do list, but there are other tasks with higher priority.

Feedback

Any kind of feedback is welcomed. Please write to info[at]pdfunit.com.

Chapter 2. Quickstart

Quickstart

Let's assume you have a some programs that generates PDF documents and you want to make sure that the programs do what they should. Further expect that your test document has exactly one page and contains the greeting “Thank you for using our services.” and the value “30.34 Euro” for the total bill. Then you check these requirements with PDFUnit as follows:

```
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_de.pdf";

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

lives_ok {
  my $expectedGreeting = "Thank you for using our services.";

  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();
```

The statements `diag $@->getMessage() if $@` makes error messages from PDFUnit-Java visible. For the intended error in the last test case the error message looks as follows:

```
# 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'.
```

That's it. The following chapters describe the features, typical test scenarios and problems when testing PDF documents.

Chapter 3. Test Scopes

3.1. Overview

An Introduction to the Syntax

All tests need the statement use `PDF::PDFUnit`. The examples use also function of `Test::Exception` and `Test::More`.

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

Each test for a **single PDF document** begins with the method `AssertThat->document(..)`. The file to be tested is passed as a parameter. Then each following function opens a different test scope, e.g. content, fonts, layouts, etc.:

The next listings give some examples for that. The `use` statement is omitted for brevity.

```
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 The PDF document can be passed to the function as a `String`, `File`, `InputStream`, `URL` or `byte array`.

It is possible to write test for a given **set of PDF documents**. Such tests start with the method `AssertThat->eachDocument(..)`:

```
# Instantiation of PDFUnit for multiple documents:
...
my $pdfArray = [$pdfUnderTest1, $pdfUnderTest2];
AssertThat->eachDocument($pdfArray)     ❷ 5: “Folders and Multiple Documents” (p. 13)
          ->hasFormat(A4_PORTRAIT)
;
...
```

❷ The PDF documents can be passed to the function as a `String[]`, `File[]`, `InputStream[]`, or `URL[]`.

A test can also cover **all PDF documents in a folder**. Such tests start with the method `AssertThat->eachDocument()->inFolder(..)`:

```
# Instantiation of PDFUnit for a folder:
...
AssertThat->eachDocument()
          ->inFolder($folder)              ❸ 5.3: “Validate all documents in a folder” (p. 14)
          ->passedFilter($allPdfunitFiles)
          -> ...
;
...
```

❸ All PDF documents in this folder will be checked. PDF documents in subfolders will not be checked.

Test Scopes

The following list gives a complete overview over the test scopes of PDFUnit. These tests are not described in this manual, to avoid redundant documentation. To read the details, follow the link to the PDFUnit-Java manual: http://www.pdfunit.com/en/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/en/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(..)
```

Sometimes, two methods are needed to enter a test scope:

```
# 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(..)
```

Once a again the link to the PDFUnit-Java manual: http://www.pdfunit.com/en/documentation/java/.

PDFUnit is continuously being improved and the manual kept up to date. Wishes and requests for new functions are appreciated. Please, send them to info[at]pdfunit.com.

Chapter 4. Comparing a Test PDF with a Reference

4.1. Overview

Many tests follow the principle of comparing a newly created test document with a PDF document which has already been validated. Such tests are useful if the process that creates the PDF has to be changed, but the output should be the same.

Initialization

The instantiation of a reference document is done with the method `->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 If the test document is password protected, the second parameter is needed.

❷ If the referenced document is not password protected, only the file name is needed. Otherwise a password has be used passed to the method.

Passwords are “only” used to open the documents. They do not influence the tests.

Overview

The following list gives a complete overview of all tests which compare two PDF files. Links after each tag refer to the chapter which describes it in detail.

The following list gives a complete overview of all tests which compare two PDF files. The details are described in the manual of PDFUnit-Java to avoid redundant documentation. Follow the link to the PDFUnit-Java manual: http://www.pdfunit.com/en/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/en/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()
```

Once again the link to the PDFUnit-Java manual: http://www.pdfunit.com/en/documentation/java/.

Chapter 5. Folders and Multiple Documents

5.1. Overview

For such tests almost all test methods are available, which exist for tests with a single PDF document. The following list shows the available methods The detailed description of each testing method can be found in the manual of PDFUnit-Java (to avoid redundant documentation). Follow this link: http://www.pdfunit.com/en/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/en/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(..)
```

A test with multiple documents or folders stops at the first detected error.

The next two chapters show tests using multiple documents and a folder.

5.2. Test Multiple PDF Documents

The following code shows how to use multiple documents in one test.

```
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";
```

The PDF documents are passed to the method `eachDocument()` as a `String[]`. Also, the types `File[]`, `InputStream[]` and `URL[]` can be used.

5.3. Validate all documents in a folder

All tests with folders start with the following methods:

```
# Instantiation of PDFUnit for a folder:

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

❶❷ These two methods start each test using all files in the given folder that have the extension '.pdf'. This extension is case-insensitive.

❸ An optional filter can be used to reduce the number of documents. Multiple filters can be concatenated. Without a filter, all PDF documents in the folder will be checked.

A test ends for all documents when one document in the folder fails.

If the filters filter out every document, an error message is shown. Also, an error message will be shown if a folder contains no PDF documents.

Example - Validate all Documents in a Folder

The following example checks whether all PDF documents in the given folder comply with the ZUGFeRD specification version 1.0:

```
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";
```

Example - Filter Out Documents by Name

In the next example, all PDF documents with 'doc-under-test' or 'reference' in their names are selected. Then the title is validated in each of the selected documents.

```
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";
```

There are two kinds of filters. The filter `FilenameContainingFilter` filters files whoes names (including the path) **contain** the given substring. The filter `FilenameMatchingFilter` evaluates whether names **matches** a regular expression. The regular expression should always begin with '`.*`'.

Chapter 6. Unicode

PDF Documents Containing Unicode

Would the tests described so far also run with content that is not ISO-8859-1, for example with Russian, Greek or Chinese text?

A difficult question. A lot of internal tests are done with Greek, Russian and Chinese documents, but tests are missing for Hebrew and Japanese documents. All in all it is not 100% clear that every available test will work with every language, but it should.

When you need to process Unicode data, it is good practice to configure all your tools to UTF-8.

The following hints may solve problems not only when working with UTF-8 files under PDFUnit. They may also be helpful in other situations.

Single Unicode Characters

Metadata and keywords can contain Unicode characters. If your operating system does not support fonts for foreign languages, you can use Unicode escape sequences in the format `\x{nnnn}` within double quoted strings. For example the copyright character “©” has the Unicode sequence `\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";
```

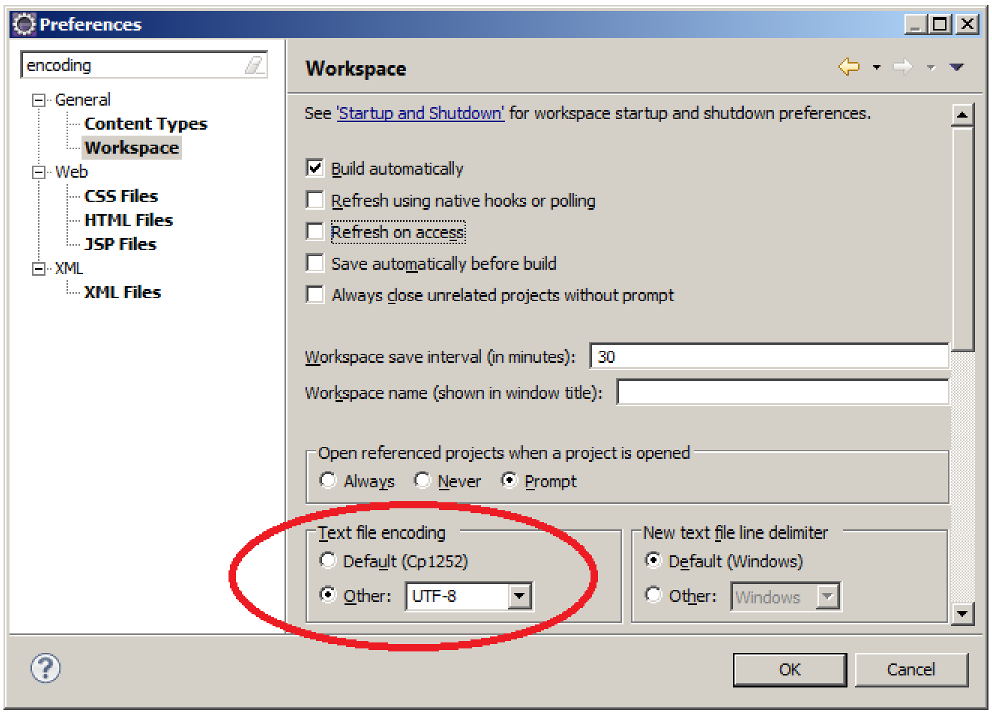
Longer Unicode Text

Of course you can use Unicode inside Perl code. Don't forget the statement `use utf8;`. A test with a longer sequence may look like this:

```
#
#  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";
```

Configure Eclipse to UTF-8

When you are working with XML files in Eclipse, you do not need to configure Eclipse for UTF-8, because UTF-8 is the default for XML files. But the default encoding for other file types is the encoding of the file system. So it is recommended to set the encoding for the entire workspace to UTF-8:



Preferences

encoding

- General
  - Content Types
  - Workspace
- Web
  - CSS Files
  - HTML Files
  - JSP Files
- XML
  - XML Files

Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

[✓] Build automatically
[ ] Refresh using native hooks or polling
[ ] Refresh on access
[ ] Save automatically before build
[ ] Always close unrelated projects without prompt

Workspace save interval (in minutes): 30

Workspace name (shown in window title):

Open referenced projects when a project is opened: Always Never [● Prompt]

Text file encoding: Default (Cp1252) [● Other: UTF-8]

New text file line delimiter: [● Default (Windows)] Other: Windows

This default can be changed for each file.

Unicode for invisible Characters - &nbsp;

A problem can occur due to a “non-breaking space”. Because at first it looks like a normal space, the comparison with a space fails. But when using the Unicode sequence of the “non-breaking space” (`\u00A0`) the test runs successfully. Here's the test:

```
#
# 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)";
```

Chapter 7. Utility Programs

7.1. General Remarks for all Utilities

PDFUnit provides utility programs to extract several parts of a PDF document into separate files, which can then be used in tests. These programs are not described in this manual, to avoid redundant documentation. Follow the link to the PDFUnit-Java manual http://www.pdfunit.com/en/documentation/java/ to read the details. The following utilities are available:

```
# 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/en/documentation/java/
#

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

The utility programs generate files. Their names are derived from those of the input files. The following rules are used to avoid naming conflicts with existing files:

- Generated file names start with an underscore.
- The names have two suffices. The penultimate is `.out` and the last one is the typical suffix for the kind of file type.

For example, when you extract bookmarks from `foo.pdf`, the file `_bookmarks_foo.out.xml` is created. Rename it before using it in a test, because then it is no longer an output file.

The Windows batch scripts in the following chapters demonstrate how to start the programs. These scripts are part of the PDFUnit release, but you have to adapt most of their content to your environment anyway: you need to set the classpath, input file and output directory.

When you start a program without parameters or with incorrect parameters, PDFUnit shows a message detailing the corect command line parameters.

Chapter 8. Installation, Configuration, Update

8.1. Technical Requirements, Java

The installations of PDF::PDFUnit follows the way of all CPAN moduls. You can find the documentation on http://search.cpan.org/dist/PDF-PDFUnit/. In addition, the file README contains the same description.

PDF::PDFUnit uses PDFUnit-Java. To avoid redundant documentation, this chapter points to the PDFUnit-Java manual (http://www.pdfunit.com/en/documentation/java/). Look for chapter 'Installation, Configuration, Update'.

Chapter 9. PDFUnit for Non-Perl Systems

9.1. A quick Look at PDFUnit-Java

The first implementation of PDFUnit was “PDFUnit-Java”. It is the reference for implementations in other programming languages. Whenever it is possible, the keywords in all implementations are chosen to be the same as in PDFUnit-Java.

The following examples shows that the API follows the “Fluent Interface” (http://de.wikipedia.org/wiki/Fluent_Interface):

```
@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)
  ;
}
```

A detailed documentation of PDFUnit-Java is available from .

9.2. A quick Look at PDFUnit-XML

It is unnecessary for testers to know Java to write tests for PDF documents. With the name 'PDFUnit-XML' a version of PDFUnit exists for an XML-based system. It contains a runtime to execute the tests, scripts to start them, XML Schema to validate the tests and stylesheets as part of the runtime. 'PDFUnit-XML' is completely compatible with 'PDFUnit-Java'.

The following examples show the idea behind 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>
```

Names of the tags and attributes are mostly the same as the function names in the Java-API. They also follow the idea of 'Fluent Interfaces' (http://de.wikipedia.org/wiki/Fluent_Interface).

XML Schema exists to validate the XML syntax.

A detailed description of PDFUnit-XML is available.

9.3. A quick Look at PDFUnit-NET

A 'PDFUnit-NET' version for a .NET environment is provided since December 2015.

```
[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")
  ;
}
```

PDFUnit-NET is fully compatible to PDFUnit-Java because a DLL is generated from the Java version. However, this means that method names in C# begin with lowercase letters.

PDFUnit-NET comes with an own manual.

Chapter 10. Typical Errors

10.1. Typical Errors, Overview

Every programmer makes errors. Many programmers repeat the same errors. Those, who knows errors, find bugs faster. This chapter informs you about typical errors and their solutions.

Overview of described errors:

```
# Overview of typical errors:

10.2: “File not Found” (p. 22)
10.3: “Java Syntax Error due to typing error” (p. 22)
10.4: “Incorrect Use of a Constructor” (p. 23)
10.5: “Use Arrays for Java-varargs” (p. 23)
10.6: “One Parameter is 'null'” (p. 24)
10.7: “Java-” (p. 24)
```

10.2. File not Found

Error Message

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

Explanation

PDF documents and other files have to be found by the Java process (PDFUnit-Java) in the classpath.

Code with Error

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

Good Code

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

10.3. Java Syntax Error due to typing error

Error Message

```
# 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.
```

Explanation

The modul Inline::Java searches the method with the wrong name in all known Java classes, can't found it and throws an exception. The error message contains the name of the analyzed class. Read the Javadoc documentation, maybe you find the correct written method.

Code with Error

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

Good Code

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

10.4. Incorrect Use of a Constructor

Error Message

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

Explanation

To invoke the constructor of a Java class the function new has to be used. Otherwise this error occurs.

Code with Error

```
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";
```

Good Code

```
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. Use Arrays for Java-varargs

Error Message

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

Explanation

A Java method may have a variable number of parameters. Perl must pass an array to such methods.

Code with Error

```
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";
```

Good Code

```
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. One Parameter is 'null'

Error Message

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

Explanation

Maybe, the assignment to a variable was forgotten or was lost during a code change. PDFUnit-Java detects null-parameters and throws this exception.

Code with Error

```
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";
```

Good Code

```
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-“point” instead of Perl-“arrow”

Error Message

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

Explanation

This error may occur when you copy a code snippet from the Java documentation and forget to change it into Perl syntax. Change the 'point' of Java into the 'arrow' of Perl.

Code with Error

```
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";
```

Good Code

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

Chapter 11. Appendix

11.1. Instantiation of PDF Documents

Again it should be pointed to the manual of PDFUnit-Java (http://www.pdfunit.com/en/documentation/java/). In its appendix different aspects around testing PDF documents are explained. It would be redundant to write it down here again. Because the names of methods and constants are the same in Java and in Perl, it should be easy for Perl programmers to transfer the Java examples to Perl.

The following topics are described in the manual of PDFUnit-Java:

```
# Contents of the appendix of the documentation of PDFUnit-Java
#
# see http://www.pdfunit.com/en/documentation/java/
#

- Instantiation of PDF Documents
- Page Selection
- Defining Page Areas
- Comparing Text
- Whitespace Processing
- Single and Double Quote Marks inside Strings
- Date Resolution
- Format Units - Points and Millimeters
- Error Messages
- Set Language for Error Messages
- Using XPath
- JAXP-Configuration
- Version History
- Unimplemented Features, Known Bugs
```

Index