

Mustang Project Nutzerdokumentation

Jochen Stärk

zu Mustangproject 1.5.1, 13.01.2018

<http://www.mustangproject.org>

Inhaltsverzeichnis

Mustang Project Nutzerdokumentation.....	1
Über Mustangproject.....	1
Übersicht von ZUGFeRD-Lösungen.....	1
Download/Projekteinrichtung.....	2
Source code.....	2
Projekteinrichtung ohne Maven.....	2
Mit Maven.....	3
Lesen von ZUGFeRD-Daten.....	3
Komplettes Lesebeispiel.....	4
Schreiben einer ZUGFeRD-PDF-Datei.....	4
Komplettes Schreibbeispiel.....	7
Schreiben eigener XML-Daten.....	7
Zusatzfunktionen.....	8

Über Mustangproject

Mustangproject ist eine Java-Bibliothek zur Unterstützung von erweiterten („ZUGFeRD“-)Metadaten in PDF-Rechnungsdateien. Sie verwendet als Eingabe PDF/A-Dateien, benötigt die Apache PDFBox-Bibliothek und unterliegt wie diese der APL-Lizenz. Sie ist daher, entsprechend den Richtlinien der Apache Public License, unter Einbettung einer entsprechenden „Notice“-Datei kostenlos einsetzbar in kommerziellen und nichtkommerziellen Projekten.

Übersicht von ZUGFeRD-Lösungen

	Plattform	Lizenz	ZF Versionen	Funktionsumfang				Geeignet für			Preis
				Lesen	XML erzeugen	PDF Schreiben	PDF/A-Umwandlung	Kommerz. Software	Freeware	Open Source	
intarsys	Java	proprietär	1	Ja	Ja	Ja	Ja	Ja	Ja	Nein	a.A.
Konik	Java	AGPL	1	Ja	Ja	Ja	Nein	Nein	Nein	Ja	0 €
Musta	Java	APL	1	Ja	Ja	Ja	Nein	Ja	Ja	Ja	0 €

ng											
https://github.com/akretion/factur-x	Python	BSD	1,2	Nein	Nein	Ja	Nein	Ja	Ja	Ja	0€
https://github.com/stephans-tapel/ZUGFeRD-csharp	C#	APL	1	Ja	Ja	Nein	Nein	Ja	Ja	Ja	0 €

Download/Projekteinrichtung

Source code

Heimat der Mustangprojekt-Quelltexte ist <https://github.com/ZUGFeRD/mustangproject/>

Projekteinrichtung ohne Maven

Mit installiertem OpenOffice.org oder LibreOffice und Eclipse for Java.

1. Starten Sie Eclipse, Neues Java-Eclipse-Projekt erstellen, beispielsweise „MustangSample“.
2. Wechseln Sie in der Shell in den erstellten Ordner.
3. Download von
 1. Mustang
 1. Downloaden Sie <http://mustangproject.org/deploy/mustang-1.5.1.jar>¹
 2. Downloaden Sie <http://mustangproject.org/deploy/NOTICE>
 2. Laden Sie
 1. Direkt eine PDF-A/1-Musterrechnung noch ohne ZUGFeRD-MustangGnuaccountingBeispielRE-2017etadaten von http://www.mustangproject.org/MustangGnuaccountingBeispielRE-20171118_506blanko.pdf herunter.
 2. oder erzeugen sich alternativ
 1. ein eigenes Beispiel-PDF durch Download der Quelldatei im OpenOffice.org-Format http://www.mustangproject.org/MustangGnuaccountingBeispielRE-20171118_506.odt und

¹ Falls Sie ohnehin PDFBox verwenden (pdfbox, fontbox, preflight, xmpbox sowie deren Abhängigkeiten apache-commons-io und apache-commons-logging) können Sie auch die wesentlich kleinere <http://mustangproject.org/deploy/original-mustang-1.5.1.jar> verwenden

1. Öffnen Sie die Datei in OpenOffice.org
2. Datei|Exportieren als PDF: Wichtig ist hier, dass Sie die Checkbox PDF/A-1a setzen
3. Speichern Sie die PDF-Datei beispielsweise als „[MustangGnuaccountingBeispielRE-20171118_506blanko.pdf](http://www.mustangproject.org/MustangGnuaccountingBeispielRE-20171118_506blanko.pdf)“ im MustangSample-Ordner
4. Wechseln Sie zurück zu Eclipse. Fügen Sie durch Rechtsklick auf das Projekt („Eigenschaften“) die heruntergeladene JAR-Dateien Projekteigenschaften als „externe Jars“ zum „Java Build Path“ Reiter „Bibliotheken“ hinzu.

Mit Maven

Mit folgendem Repository

```
<repositories>
  <repository>
    <id>mustang-mvn-repo</id>
    <url>https://raw.githubusercontent.com/ZUGFeRD/mustangproject/mvn-repo</url>
  </repository>
</repositories>
```

und folgender Dependency

```
<dependency>
  <groupId>org.mustangproject.ZUGFeRD</groupId>
  <artifactId>mustang</artifactId>
  <version>1.5.1</version>
</dependency>
```

Bei der Gelegenheit kann man auch gleich Apache Commons Logging und JAXB importieren:

```
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.1.1</version>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.2.5</version>
</dependency>
```

Lesen von ZUGFeRD-Daten

1. Laden Sie sich eine PDF-Datei mit ZUGFeRD-Daten herunter, beispielsweise http://www.mustangproject.org/MustangGnuaccountingBeispielRE-20171118_506.pdf
2. Erstellen Sie eine neue Java-Klasse unterhalb von src, beispielsweise MustangReader. inklusive „Public static void main()“
3. Geben Sie innerhalb von Main „ZUGFeRDImporter zi=new ZUGFeRDImporter();“ ein und lassen Sie den Import durch STRG+SHIFT+O ergänzen
4. verwenden Sie zi.extract(PDF-Dateiname) und ggf. canParse() um festzustellen ob es sich um

ZUGFeRD-Daten handelt.

5. Nach `zi.parse()` haben Sie Zugriff auf die getter wie `getAmount()`
6. Welche Daten enthalten sind, können Sie der XML-Datei entnehmen die im ZUGFeRD-Beispiel-PDF eingebettet ist

Komplettes Lesebeispiel

```
package sample;

import org.mustangproject.ZUGFeRD.ZUGFeRDImporter;

public class Read {

    public static void main(String[] args) {
        ZUGFeRDImporter zi=new ZUGFeRDImporter();
        zi.extract("./MustangGnuaccountingBeispielRE-20171118_506.pdf");
        System.out.println("Lese ZUGFeRD");
        if (zi.canParse()) {
            zi.parse();
            System.out.println("Fälliger Betrag:"+zi.getAmount());
            System.out.println("BIC:"+zi.getBIC());
            System.out.println("IBAN:"+zi.getIBAN());
            System.out.println("Kontoinhaber:"+zi.getHolder());
            System.out.println("Rechnungsnr:"+zi.getForeignReference());
        }

    }

}
```

Schreiben einer ZUGFeRD-PDF-Datei

Ein Beispielprogramm zum Schreiben ist deshalb umfangreicher, weil erstens mehr Daten in einer differenzierteren Struktur geschrieben werden als derzeit beim Lesen benötigt und zweitens dem Exporter die Daten per Interface zur Verfügung gestellt werden müssen.

1. Erstellen Sie eine neue Klasse unterhalb von `src`, beispielsweise `MustangWriter` inklusive des obligatorischen „Public static void main()“ .
2. Ändern Sie `public class MustangWriter` in `public class MustangWriter implements IZUGFeRDExportableTransaction`
3. Fügen Sie innerhalb der Klasse `MustangWriter` folgende Klassen hinzu
 1. `class Contact implements IZUGFeRDExportableContact {}`
 2. `class Item implements IZUGFeRDExportableItem {`
 `private BigDecimal price, quantity;`
 `private Product product;`
 `}`
 3. `class Product implements IZUGFeRDExportableProduct {`
 `private String description, name, unit;`
 `private BigDecimal VATPercent;`
 `}`

4. Generieren Sie die Imports durch Drücken von STRG+SHIFT+O
5. Markieren Sie den Klassennamen MustangWriter und drücken Sie ALT+SHIFT+S, wählen Sie Override/Implement Methods und drücken Return.
6. Klicken Sie auf Contact und Wiederholen Sie den letzten Schritt.
7. Klicken Sie auf Item, markieren Sie die Variablen und wählen Sie zuerst „Generate Getters and Setters“ nach drücken von ALT+SHIFT+S. Wählen Sie alle Member aus und drücken Sie Return.
8. Klicken Sie erneut auf Item, drücken von ALT+SHIFT+S und wählen „Generate Constructor using Fields“. Wählen Sie erneut alle Member aus und drücken Sie Return.
9. Auf Item muss der Quick Fix „add unimplemented methods“ ausgeführt werden, die zwei generierten Funktionen (getItemAllowances und getItemCharges) dürfen null zurück geben.
10. Wenden Sie die beiden letzten Schritte auch auf Product an: Klicken Sie auf Product, markieren Sie die Variablen und wählen Sie „Generate Getters and Setters“ nach drücken von ALT+SHIFT+S. Wählen Sie alle Member aus und drücken Sie Return.
11. Item benötigt neben den getter/setter auch noch andere Methode, wählen Sie Item aus, drücken Sie ALT+SHIFT+S, wählen Sie Override/Implement Methods
12. Klicken Sie erneut auf Product, drücken von ALT+SHIFT+S und wählen „Generate Constructor using Fields“. Wählen Sie erneut alle Member aus und drücken Sie Return.
13. Folgende Methoden von Contact sollten Folgendes zurückgeben:

```

1. getCountry(): "DE"
2. getLocation(): "Spielkreis"
3. getName(): "Theodor Est"
4. getStreet(): "Bahnstr. 42"
5. getVATID(): "DE999999999"
6. getZIP(): "88802";

```

14. Folgende Methoden der Hauptklasse -MustangWriter- sollten folgendes zurückgeben:

```

1. getDeliveryDate(): new
   GregorianCalendar(2017, Calendar.NOVEMBER, 17).getTime()
2. Zweimaliges CTRL+SHIFT+O importiert die dazu nötige GregorianCalendar und Calendar
   Klasse
3. getDueDate(): new GregorianCalendar(2017, Calendar.DECEMBER, 9).getTime()
4. getIssueDate(): new GregorianCalendar(2017, Calendar.NOVEMBER, 18).getTime()
5. getNumber(): "RE-20171118/506"
6. getOwnBIC(): "COBADEFFXXX"
7. getOwnBankName(): "Commerzbank"
8. getOwnCountry() "DE"
9. getOwnIBAN(): "DE88 2008 0000 0970 3757 00"
10. getOwnLocation() "Stadthausen"
11. getOwnOrganisationName(): "Bei Spiel GmbH"
12. getOwnStreet() "Ecke 12"
13. getOwnTaxID(): "22/815/0815/4"
14. getOwnVATID(): "DE136695976"
15. getOwnZIP(): "12345"

```

```

16.    getOwnOrganisationFullPlaintextInfo(): "Bei Spiel GmbH\n"+
        "Ecke 12\n"+
        "12345 Stadthausen\n"+
        "Geschäftsführer: Max Mustermann"
17.    getRecipient(): new Contact()

```

18. getZFItems() der Hauptklasse kann jetzt Produkte anlegen und diese als Array von Posten (Items) zurückliefern:

```

        Item[] allItems=new Item[3];
        Product designProduct=new Product("", "Künstlerische Gestaltung
(Stunde): Einer Beispielrechnung", "HUR", new BigDecimal("7.000000"));
        Product balloonProduct=new Product("", "Luftballon: Bunt, ca. 500ml",
"C62", new BigDecimal("19.000000"));
        Product airProduct=new Product("", "Heiße Luft pro Liter", "LTR", new
BigDecimal("19.000000"));

        allItems[0]=new Item(new BigDecimal("160"), new BigDecimal("1"),
designProduct);
        allItems[1]=new Item(new BigDecimal("0.79"), new BigDecimal("400"),
balloonProduct);
        allItems[2]=new Item(new BigDecimal("0.10"), new BigDecimal("200"),
airProduct);
        return allItems;

```

19. Legen Sie eine neue Funktion, beispielsweise „apply“ an (private reicht).
20. In der Main-Methode der Hauptklasse instantiiert man jetzt die Klasse und rufen Sie dort apply() auf.
21. In der apply-Methode kann man jetzt
1. eine ZUGFeRDEXporterFactory instantiiieren,
 2. darauf Producer und Creator setzen (bspw. ZUGFeRDEXporter ze=new ZUGFeRDEXporterFromA1Factory().setProducer("string").setCreator("string")) und über load die PDF-A/1-Datei angeben und über den Rückgabewert den ZUGFeRDEXporter holen. Die ZUGFeRD-Version kann man in der setProducer.setCreator-Kette mit setZUGFeRDVersion(2) angeben.
 3. Auf dem Exporter ruft man diePDFAttachZugferdFile-Methode (mit this als IZUGFeRDEXportableTransaction) auf und
 4. benutzt schlussendlich die export-Funktion. Die apply-Methode sieht dann – mit entsprechenden try/catch-Blöcken- beispielsweise so aus:

```

try {
    System.out.println("Lese Blanko-PDF");
    ZUGFeRDEXporter ze = new
ZUGFeRDEXporterFromA1Factory().setProducer("My
Application").setCreator(System.getProperty("user.name")).load("./MustangGnuaccount
ingBeispielRE-20171118_506blanko.pdf");
    System.out.println("Generiere ZUGFeRD-Daten");
    ze.PDFAttachZugferdFile(this);
    System.out.println("Schreibe ZUGFeRD-PDF");
    ze.export("./MustangGnuaccountingBeispielRE-

```

```

20171118_506new.pdf");
    System.out.println("Fertig.");
} catch (IOException e) {
    e.printStackTrace();
}

```

22. Ein CTRL+SHIFT+O hilft wieder beim Hinzufügen der nötigen Imports
23. „My Application“ und `System.getProperty("user.name")` werden in den Metadaten als „Producer“ (in etwa: erstellende Anwendung) beziehungsweise „Creator“ (in etwa: Autor) gespeichert. Bitte passen Sie die Werte entsprechend Ihrer Anwendung an.
24. Starten Sie, es sollte eine valide ZUGFeRD-Rechnung in `./MustangGnuaccountingBeispielRE-20171118_506new.pdf` erstellt werden.
25. Passen Sie ggf. die NOTICE-Datei an und fügen Sie Ihrer Anwendung hinzu.
26. Stellen Sie sicher, dass die XML-Daten in der dem PDF eingebetteten ZUGFeRD-invoice.xml immer den menschenlesbaren Daten im PDF entsprechen.

Die erzeugte Datei beinhaltet ZUGFeRD-invoice.xml statt factur-x.xml im offiziellen Beispiel so lange die ZUGFeRD-Version in der Factory nicht auf 2 gesetzt wurde.

Komplettes Schreibbeispiel

Siehe MustangWriter.java in diesem Verzeichnis.

Schreiben eigener XML-Daten

Sollten Sie eine eigene Implementierung verwenden um ZUGFeRD-XML-Daten zu erzeugen können Sie diese mit `setZUGFeRDXMLData` schreiben:

```

ZUGFeRDExporter ze;
try {

    System.out.println("Konvertiere zu PDF/A-3u");
    ze = new
ZUGFeRDExporterFromAlFactory().setProducer("My
Application").setCreator(System.getProperty("user.name")).load("./MustangGnuaccountingBeispielRE-
20171118_506blanko.pdf");

    System.out.println("Hänge ZUGFeRD-Datei an");
    String ownZUGFeRDXML = "<rsm:CrossIndustryDocument></rsm:CrossIndustryDocument>";
    ze.setZUGFeRDXMLData(ownZUGFeRDXML.getBytes());
    System.out.println("Schreibe ZUGFeRD-PDF");
    ze.export("./Target.pdf");
    System.out.println("Fertig.");

} catch (IOException e) {
    e.printStackTrace();
}

```

Mustangproject prüft, die Eingabedatei auf einigermaßen korrektes PDF/A ist und dass die XML-Daten mindestens `<rsm:CrossIndustry` enthalten was auf ZF1- (CrossIndustryDocument) und ZF2-Daten (CrossIndustryInvoice) zutrifft.

Zusatzfunktionen

- `ZUGFeRDExporter.setTest()` setzt ein Attribut im ZUGFeRD-XML, das benutzt wird um eine Testrechnung auszuzeichnen.
- `ZUGFeRDExporter.ignoreAllErrors()` überspringt die Überprüfung der Eingangsdatei auf PDF/A-1-Fehler
- Ein erster Versuch bestehendes ZUGFeRD1-XML in 2 zu konvertieren kann mit `String facturx=new ZUGFeRDMigrator().migrateFromV1ToV2(zugferdInvoice);` begonnen werden.